N° d'enregistrement
au C.N.R.S. :

THESE

PRESENTEE A

L'UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

POUR OBTENIR

LE TITRE DE DOCTEUR INGENIEUR

PAR

Michel BARON

INGENIEUR DE L'ARMEMENT

Sujet de la thèse

DOCUMENTATION ET HOMOLOGATION DES PROGRAMMES DE GESTION A L'AIDE DE TABLES DE DECISIONS

Soutenue le 8 décembre 1973 devant la Commission d'Examen

Monsieur le Professeur	*L. BOLLIET*	*Président*
Messieurs	*J.M. DETHOOR*	*Rapporteur*
	C. DELOBEL	*Examinateurs*
	J.P. CRESTIN	

N° d'enregistrement
au C.N.R.S. :

THESE

PRESENTEE A

L'UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

POUR OBTENIR

LE TITRE DE DOCTEUR INGENIEUR

PAR

Michel BARON

INGENIEUR DE L'ARMEMENT

Sujet de la thèse

DOCUMENTATION ET HOMOLOGATION DES PROGRAMMES DE GESTION A L'AIDE DE TABLES DE DECISIONS

Soutenue le 8 décembre 1973 devant la Commission d'Examen

Monsieur le Professeur	*L. BOLLIET*	*Président*
Messieurs	*J.M. DETHOOR*	*Rapporteur*
	C. DELOBEL	*Examinateurs*
	J.P. CRESTIN	

UNIV.
LEIDEN
BIBL.

Président : Monsieur Michel SOUTIF
Vice-Président : Monsieur Gabriel CAU

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Georges	Clinique des maladies infectieuses
	ARNAUD Paul	Chimie
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BEAUDOING André	Pédiatrie
	BENOIT Jean	Radioélectricité
	BERNARD Alain	Mathématiques Pures
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BESSON Jean	Electrochimie
	BEZES Henri	Chirurgie générale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNETAIN Lucien	Chimie minérale
	BONNET-EYMARD Joseph	Pathologie médicale
	BONNIER Etienne	Electrochimie Electrométallurgie
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques Appliquées
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du Solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Guy	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et Toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHATEAU Robert	Thérapeutique
	CHENE Marcel	Chimie papetière
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie Pathologique
	COUMES André	Radioélectricité
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie

Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41544309_0031

	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
MM.	DODU Jacques	Mécanique appliquée
	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de Dermatologie et Syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	FELICI Noël	Electrostatique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures
	GASTINEL Noël	Analyse numérique
	GAVEND Michel	Pharmacologie
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques Pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	KAHANE André	Physique générale
	KLEIN Joseph	Mathématiques Pures
	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques Appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre	Mathématiques Appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques Pures
	MALGRANGE Bernard	Mathématiques Pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEEL Louis	Physique du Solide
	OZENDA Paul	Botanique
	PAUTHENET René	Electrotechnique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET René	Servomécanismes
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique Industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	DE ROUGEMONT Jacques	Neurologie
	SANTON Lucien	Mécanique

MM.	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SILBERT Robert	Mécanique des fluides
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique Nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
M.	VERAIN André	Physique
MM.	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	CHEEKE John	Thermodynamique
	GILLESPIE John	I. S. N.
	ROCKAFELLAR Ralph	Mathématiques appliquées
	WOHLFARTH Erich	Physique du solide

PROFESSEURS SANS CHAIRE

MM.	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
	BERTRANDIAS Jean-Paul	Mathématiques appliquées
	BIAREZ Jean-Pierre	Mécanique
Mme	BONNIER Jane	Chimie générale
MM.	CARLIER Georges	Biologie végétale
	COHEN Joseph	Electrotechnique
	DEPASSEL Roger	Mécanique des Fluides
	DURAND Francis	Métallurgie
	GAUTHIER Yves	Sciences biologiques
	GAUTRON René	Chimie
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	HACQUES Gérard	Calcul numérique
	IDELMAN Simon	Physiologie animale
	JANIN Bernard	Géographie
	JOLY Jean-René	Mathématiques Pures
	JULLIEN Pierre	Mathématiques appliquées
Mme	KAHANE Josette	Physique
MM.	MAYNARD Roger	Physique du solide
	MULLER Jean-Michel	Thérapeutique
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	PFISTER Jean-Claude	Physique du solide
	PIERY Yvette	Physiologie animale
	POULOUJADOFF Michel	Electrotechnique
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction Mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale

MM.	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques appliquées
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

Mle	AGNIUS-ĐELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Dermatologie
	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Yves	Chimie
	BEGUIN Claude	Chimie organique
	BILLET Jean	Géographie
	BLIMAN Samuel	Electronique (EIE)
	BLOCH Daniel	Electrotechnique
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BOUVARD Maurice	Mécanique des Fluides
	BRODEAU François	Mathématiques (IUT B)
	BRUGEL Lucien	Energétique
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CONTE René	Physique
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	CYROT Michel	Physique du solide
	DELOBEL Claude	M. I. A. G.
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FONTAINE Jean-Marc	Mathématiques Pures
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques Appliquées
	GROULADE Joseph	Biochimie médicale
	GUITTON Jacques	Chimie
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Médecine préventive
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	JOUBERT Jean-Claude	Physique du Solide
	KRAKOWIAK Sacha	Mathématiques appliquées
	KUHN Gérard	Physique
	LACOUME Jean-Louis	Physique
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LANCIA Roland	Physique atomique
	LE JUNTER Noël	Electronique
	LEROY Philippe	Mathématiques
	LOISEAUX Jean-Marie	Physique Nucléaire
	LUU DUC Cuong	Chimie Organique
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et Médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)

MM.	MICHOULIER Jean	Physique (I. U. T. "A")
	MICOUD Max	Maladies infectieuses
	MOREAU René	Hydraulique (INP)
	NEGRE Robert	Mécanique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PELMONT Jean	Physiologie animale
	PERRET Jean	Neurologie
	PERRIN Louis	Pathologie expérimentale
	PHELIP Xavier	Rhumatologie
	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RAYNAUD Hervé	M. I. A. G.
	RENAUD Maurice	Chimie
	RICHARD Lucien	Botanique
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROMIER Guy	Mathématiques (IUT B)
	SHOM Jean Claude	Chimie générale
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VEILLON Gérard	Mathématiques appliquées (INP)
	VOOG Robert	Médecine interne
	VROUSSOS Constantin	Radiologie

MAITRES DE CONFERENCES ASSOCIES

MM.	CRABLEE Pierre	C. E. R. M. O.
	CURRIE Jan	Mathématiques appliquées
	YACOUD Mahmoud	Médecine légale

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

Mme	BERIEL Hélène	Physiologie
Mme	RENAUDET Jacqueline	Microbiologie

L'idée n'est rien et, en somme, ne coûte rien. C'est le faire qui compte et faire est le propre de la main.

Paul Valéry

A ma femme et mes parents,
A mes camarades et mes maîtres,
qui ont accompagné et dirigé vingt années scolaires

TABLE DES MATIERES

CHAPITRE 1 - INTRODUCTION DEFINITION DU SUJET 1

CHAPITRE 2 - PRATIQUE DES TABLES DE DECISIONS 5

2.1 - HISTORIQUE DES TABLES DE DECISIONS 5

2.2 - DEFINITIONS GENERALES 6

2.2.1 - Structure de tables de décisions 7
2.2.2 - Définitions des termes relatifs aux tables de décisions 8

2.3 - PROPRIETES DES TABLES DE DECISIONS 9

2.3.1 - Conditions 9
2.3.2 - Actions 12
2.3.3 - Redondance et contradiction 13
2.3.4 - Table complète - Table réductible 14

2.4 - PRATIQUE DES TABLES DE DECISIONS - ANALYSE DES SYSTEMES 15

2.5 - CONCLUSION 16

CHAPITRE 3 - DEFINITIONS RELATIVES AUX TABLES A ENTREES LIMITEES 17

3.1 -POSITION DU PROBLEME - NOTATIONS 17

3.1.1 - Rappels et définitions 17
3.1.2 - Programme abstrait 19
3.1.3 - Titre d'un sous-graphe 20

3.2 - DEFINITION DES TABLES DE DECISIONS A ENTREES LIMITEES 24

3.3 - TERMES USUELS RELATIFS AUX TABLES DE DECISIONS 25

3.3.1 - Aïeul - Ruptures de séquences 25
3.3.2 - Règles - Souches - Entrées 26
3.3.3 - Propriétés 27

3.4 - TABLES DE DECISIONS CLASSIQUES - DISCUSSION DE LA DEFINITION 28

3.4.1 - 28
3.4.2 - 33
3.4.3 - Conclusion 35

CHAPITRE 4 - CONSTRUCTIONS DES TABLES A ENTREES LIMITEES D'UN PROGRAMME COBOL 37

4.1 - PRESENTATION - TABLE DE DECISIONS ABSTRAITE 37

4.2 - FORME CODEE DES TABLES DE DECISIONS 38

4.2.1 - Critères de choix d'une forme de codage 38
4.2.2 - Définition du codage 41
4.2.3 - Propriétés de la forme de codage 48
4.2.4 - Forme normale de Bacchus 52

4.3 - ELABORATION DES TABLES DE DECISIONS A ENTREES LIMITEES D'UN PROGRAMME COBOL 53

4.3.1 - Recherche des sous-graphes à structures de tables - Tables atomiques 54
4.3.2 - Graphe du programme 55
4.3.3 - Edition des tables de décisions 58

4.4 - CONCLUSION 65

CHAPITRE 5 - TABLES DE DECISIONS A ENTREES ETENDUES 67

5.1 - DEFINITION ET REPRESENTATION DES TABLES DE DECISIONS DES TABLES DE DECISIONS A ENTREES ETENDUES 68

5.1.1 - Sous-tables titrées 68
5.1.2 - Définition d'une table de décisions à entrées étendues 70
5.1.3 - Forme de codage des tables à entrées étendues 73

5.2 - CONSTRUCTION DES TABLES DE DECISIONS A ENTREES ETENDUES 76

5.2.1 - Entrées actions étendues 77
5.2.2 - Entrées conditions étendues 78

5.3 - EDITION DES TABLES DE DECISIONS A ENTREES ETENDUES 90

CHAPITRE 6 - HOMOLOGATION DES PROGRAMMES DE GESTION A L'AIDE DE TABLES DE DECISIONS 93

6.1 - PRESENTATION 93

6.2 - MODULES DE TESTS 95

6.3 - JEUX D'ESSAIS - SEMANTIQUE DES TRAITEMENTS 95

6.3.1 - Suppression des ruptures sémantiques 96
6.3.2 - Boucles de traitements 100

6.4 - HIERARCHIE DES MODULES DE TRAITEMENTS 102

6.5 - SEMANTIQUE DES ENTREES DU PROGRAMME 107

6.6 - CONCLUSION 108

CHAPITRE 7 - UTILISATION DE LA CHAINE TABOL 109

7.1 - CREATION DES FICHIERS 110

7.2 - APPEL DES PROGRAMMES D'ANALYSE SYNTAXIQUE 111

7.3 - APPEL DES PHASES PD1C ET PD2A 112

7.4 - APPEL DE LA PHASE PD2B 113

7.5 - EDITION DES TABLES 113

7.6 - TABLES DE DECISIONS A ENTREES MIXTES 113

7.6.1 - Syntaxe des directives 114
7.6.2 - Directive COND 115
7.6.3 - Directive ACT 118
7.6.4 - Directive SUPR 119
7.6.5 - Directive AJRS 121
7.6.6 - Directive REG 121
7.6.7 - Directive SEP 123
7.6.8 - Directive NUL 124
7.6.9 - Directive CHGT 124

7.7 - MEMENTO D'EMPLOI DE LA CHAINE TABOL 125

7.7.1 - JØB 1 126
7.7.2 - JØB 2 127

CHAPITRE 8 - PRESENTATION PRATIQUE DES PROGRAMMES DE LA CHAINE 129

8.1 - INTRODUCTION 129

8.2 - PRESENTATION DE LA PHASE 1 130

8.2.1 - Compilation de la DATA DIVISION 131bis
8.2.2 - Analyse syntaxique de la PROCEDURE DIVISION 137

8.3 - PROGRAMME PD1C 148

8.3.1 - Codage des tables atomiques 149
8.3.2 - Regroupement des tables atomiques d'un paragraphe 150

8.4 - REGROUPEMENT DES PARAGRAPHES. PROGRAMME PD2A 161

8.4.1 - Module I. Chargement des tables 161
8.4.2 - Module II. Premières tables 162
8.4.3 - Module III. Tables noeud de losange 163

8.5 - PROGRAMME PD2B. RUPTURES SEMANTIQUES 166

8.5.1 - Lecture et mise en mémoire d'une table 167
8.5.2 - Recherche des variables réceptrices 169
8.5.3 - Recherche des variables intervenant dans l'évaluation d'une condition 169
8.5.4 - Ecriture d'une table 170

8.6 - PHASE 3 172

8.6.1 - Programme PD3A 173
8.6.2 - Programme PD3B. Edition des tables 186
8.6.3 - Directives sémantiques. PD3C 191
8.6.4 - Programme PD3D. Directives de structure 202

ANNEXE 1 - ETUDE BIBLIOGRAPHIQUE 217

ANNEXE 2 - EXEMPLE D'APPLICATION DE LA CHAINE TABOL 272

PREFACE

Les tables de décisions existent. Elles existent depuis longtemps : leur forme a été fixée en 1962. On peut cependant affirmer qu'elles ne sont pas utilisées, aussi souvent que les avantages qu'elles présentent le permettent.

L'emploi des tables de décisions, dans l'étude analytique d'un problème commercial, simplifie la vérification systématique des algorithmes, assure une documentation précise et synthétique et, ce n'est pas son moindre avantage, accessible aux non-informaticiens. Par l'intermédiaire des tables de décisions, les organismes de direction pourrait, enfin, exercer leur contrôle. L'usage généralisé des tables de décisions supprimerait les effluves de soufre qui s'échappent, en volutes lourdes, de certains centres de gestion.

Quels inconvénients s'attachent donc à l'emploi des tables de décisions ? Le premier est qu'il bouleverse les habitudes acquises. Mais l'informatique a été l'objet d'évolutions plus rapides qui ont aiguisées l'esprit d'adaptation de l'utilisateur.

L'utilisation des tables de décisions ne diminue pas le travail fastidieux d'écriture et de perforation. L'existence de nombreux processeurs de tables pourrait permettre de conclure au contraire. Les tables qu'ingèrent ces précompilateurs sont formées d'instructions semblables à celles du langage de programmation. Il faudra donc écrire et perforer les instructions du programme, et générer encore ces excroissances bizarres que sont les entrées.

A travail égal, voire inférieur, un programmeur, conscient de savoir-faire, fera confiance à son expérience pour trouver approximativement, le meilleur compromis temps d'exécution - place en mémoire, de la programmation qu'il rédigera lui-même. En l'absence de toute étude relative aux tables de décisions, autres que celles qui ont conduit à la commercialisation des processeurs de tables, il renoncera donc à leur emploi.

Cependant, les avantages principaux des tables de décisions prennent leur importance après la programmation, même manuelle.

C'est l'ambition du travail que nous présentons dans cette publication, de permettre la traduction d'un programme en un ensemble de tables de décisions. Elles pourront être utilisées pour la documentation et la vérification du programme.

Ce travail a été rendu possible grâce aux conseils qu'a bien voulu donner Monsieur le professeur BOLLIET. Il a accepté de présider cette thèse. Qu'il trouve ici l'expression de ma profonde reconnaissance pour l'intérêt qu'il a porté à mon travail au cours de ces deux années.

Je suis très reconnaissant à Monsieur DELOBEL d'avoir bien voulu accepter de juger ce travail.

Monsieur l'ingénieur en chef de l'armement LAZENNEC a proposé le sujet de cette thèse. Les deux stages qu'il a autorisés au département I. C. qu'il dirige au Centre d'Electronique de l'Armement, ont permis d'ancrer dans le réel de l'informatique, les développements théoriques que nous avons pu étudier. Je remercie M. Lazennec et l'équipe du département I. C. du CELAR d'avoir bien voulu accepter de me soutenir dans ce travail.

Mes remerciements sincères vont à Monsieur DETHOOR. Sa profonde connaissance des problèmes de l'informatique, théoriques, humains et pratiques, a été un appui essentiel. Nos nombreuses et enrichissantes conversations ont largement contribué à la construction de cet ouvrage. Qu'il en soit remercié.

Je tiens à remercier Monsieur CRESTIN et le centre de calcul de l'Ecole Nationale Supérieure de Techniques Avancées ; Monsieur FRANCOIS et le service recherche, qui ont fourni le cadre studieux nécessaire à une telle étude.

Je n'oublie pas non plus tous ceux et celles qui ont contribué à la réalisation matérielle de cet ouvrage.

CHAPITRE 1

PRÉSENTATION GÉNÉRALE - DÉFINITION DU SUJET

Le sujet de l'étude soutenue dans cette thèse a été proposé par Monsieur LAZENNEC, Chef de département au Centre d'Electronique de l'Armement. La documentation automatique, ou contrôlée par l'utilisateur, de programmes de gestion écrits dans un langage évolué, n'a fait l'objet que de réalisations limitées à la documentation très technique nécessaire aux programmeurs et, par conséquent, inaccessibles à toutes les autres personnes intéressées aux programmes en cause. Nous citerons les processeurs courants de tracés d'organigrammes et l'étude menée par ECA Automation, permettant la documentation des variables d'un programme. (Références 31 à 36).

Le premier but qui nous a été proposé est l'étude et la réalisation d'un outil d'aide à la documentation des programmes de gestion, écrits en COBOL. Pour être satisfaisante, la documentation fournie doit être :

- précise et permettre la maintenance ultérieure du programme ;
- lisible et compréhensible pour toutes les personnes intéressées :

informaticiens et non informaticiens. D'évidence, l'ordinogramme, s'il est précis, l'est à l'excès, et offre un aspect trop analytique pour être efficace dans le cas de programmes de gestion. Il a donc été choisi d'éditer des tables de décisions. Les tables directement déduites de l'étude du programme sont précises. Elles traduisent, sans intervention de l'utilisateur, les concepts algorithmiques codés par le programme. Leur avantage est d'offrir une représentation géométriquement plus groupée que l'ordinogramme, et donc plus accessible à l'oeil humain.

Cette première documentation présente un caractère détaillé utile pour la maintenance, mais superflu pour une documentation générale du programme. Nous chercherons ensuite à éditer différents jeux de tables de décisions qui seront une documentation de plus en plus éloignée du détail et des intermédiaires de programmation pour faire apparaître l'organisation hiérarchique des modules à des niveaux de plus en plus élevés. Cette deuxième phase de la documentation se réalise par étapes, sous le contrôle de l'utilisateur. Le résultat en est un ensemble de documents de plus en plus synthétiques, accessibles à d'autres personnes que les seuls informaticiens auteurs du projet. Des phrases françaises indiquent en clair, dans les souches, les concepts algorithmiques qui avaient été codés dans le programme par des instructions COBOL.

Le problème soulevé par la documentation a posteriori des programmes peut paraître marginal dans de nombreux cas. Il a toujours paru préférable de réaliser un programme de gestion dans le cadre d'une étude complète,incluant l'édition de la documentation au cours de la phase d'analyse. Les aides informatiques à cette documentation a priori sont d'ailleurs nombreuses et développées. Cependant, une telle démarche, complète et systématique, quand elle est entreprise, n'exclut pas l'utilité d'une documentation directement et automatiquement déduite du seul programme. La comparaison entre les deux documentations amont et aval du programme peut permettre une première évaluation.

Enfin, si la documentation existante est incomplète ou peu satisfaisante, et le cas en est fréquent, l'existence d'un outil informatique permettant l'édition de documents directement déduits de la programmation devient nécessaire.

Les tables de décisions que nous obtiendrons n'auront pas, après édition, à être soumises à l'ordinateur, mais à l'homme. Il a ainsi été possible d'apporter des modifications à la forme classiquement retenue pour des tables qui alimentent les préprocesseurs et les générateurs de COBOL. Le chapitre 2 qui suit définit le vocabulaire classique utilisé dans les ouvrages relatifs aux tables de décisions. Le chapitre 3 propose de nouvelles définitions, adaptées à notre propos, des termes usuels : tables, règles, entrées, etc. Ces éléments définis, le chapitre 4 introduit les algorithmes qui permettent la traduction automatique d'un programme COBOL source,en un ensemble de tables de décisions à entrées limitées, liées les unes aux autres. Ce chapitre décrit les transformations successives que subit le programme.

Le chapitre 5 introduit une définition des tables de décisions à entrées mixtes et étendues. La transformation des tables précédentes, en tables plus orientées vers la documentation, ou tables à entrées étendues, se réalisera au moyen de directives de l'utilisateur. La seconde partie de ce chapitre décrit les traitements qu'engendrent pratiquement les principales directives de transformation.

Une documentation efficace obtenue, il est possible de l'utiliser aux fins d'évaluer les possibilités et les caractéristiques du programme. Une contribution à l'évaluation et l'homologation d'un programme de gestion à l'aide de tables de décisions fait l'objet du chapitre 6 de ce texte. Les difficultés rencontrées dans cette étude, l'ampleur et la complexité du sujet les laissaient prévoir, ont limité la portée pratique des conclusions que nous avons pu poser :

a) Ampleur du domaine possible des erreurs réelles d'un programme de gestion. Bien peu correspondent à des erreurs typées d'algorithmes. Une erreur d'un programme de gestion sera le plus souvent mise en lumière à l'occasion de l'exécution du programme sur des données erronées ou inattendues. En conséquence de ce fait d'expérience, nous avons refusé de chercher à formaliser la notion de programme seul pour nous intéresser à l'ensemble programme - valeurs sur lesquelles il agit. Une "erreur" est alors interprétée comme une incompatibilité entre les deux termes de cet ensemble. Nous proposons au chapitre 6 trois axes de recherche de ces incompatibilités, en utilisant les informations contenues dans les tables de décisions déduites du programme.

b) Complexité des structures que l'on rencontre dans les problèmes de gestion. Une recherche systématique des erreurs se heurte dans tous les cas réels à des contraintes pratiques : temps de mise en oeuvre, dimensions, qui rendent illusoires un contrôle complet du programme. Ainsi un jeu d'essais exhaustif sera de dimensions gigantesques et dès lors inutilisable pour la plupart des programmes réels non triviaux.

Il reste que nous proposons une méthodologie du contrôle de la qualité de la programmation proposée, axée sur la compatibilité que l'on doit trouver entre les traitements tels qu'ils sont programmés et la structure et l'organisation du système de fichiers, entrées et sorties de l'algorithme.

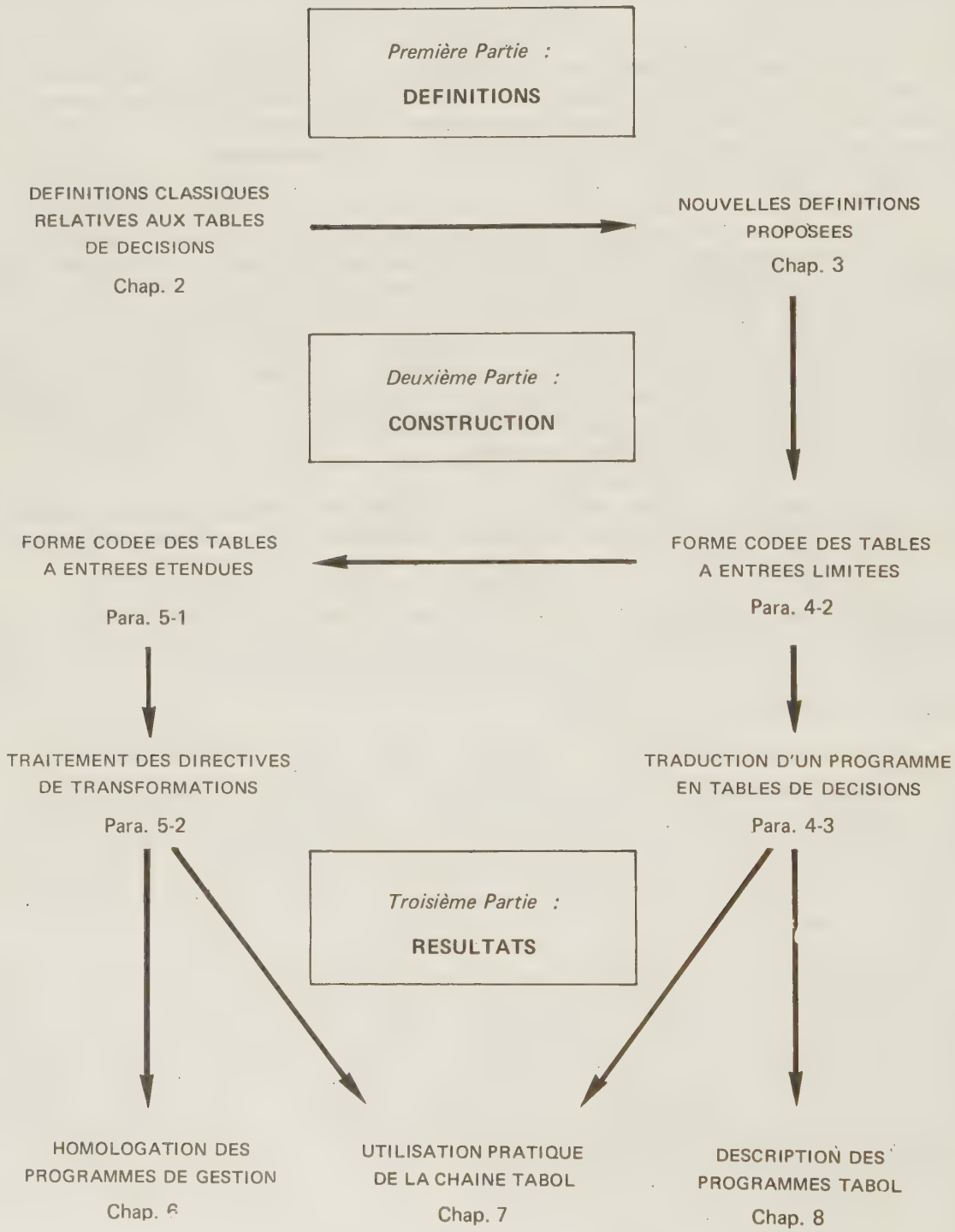
Cette méthodologie, nous l'avons montré ne prétend pas être générale. Le problème de la correction des programmes de gestion est un sujet suffisamment ouvert pour justifier des recherches plus importantes que celle-ci. L'approche que nous en faisons ici à l'aide de tables de décisions déduites du programme, apporte par son originalité même, quelques idées neuves mais partielles.

Le dernier chapitre de cet ouvrage, présente rapidement les neufs programmes de la chaîne TABOL qui réalisent l'édition des différents documents, présentés plus haut. Cette chaîne a été écrite en COBOL pour deux ordinateurs, l'un de moyenne puissance, l'autre de grande capacité.

Nous avons annexé à cette publication, d'une part une notice bibliographique et l'étude résumée de quelques articles, d'autre part les résultats de l'exécution de la chaîne TABOL sur un exemple fourni par le CELAR.

On trouvera sur la page suivante un schéma résumé de l'organisation logique des chapitres de ce texte.

DOCUMENTATION ET HOMOLOGATION DES PROGRAMMES DE GESTION A L'AIDE DE TABLES DE DECISIONS



CHAPITRE 2

PRATIQUE DES TABLES DE DÉCISIONS

2.1 - HISTORIQUE DES TABLES DE DECISIONS

L'histoire des sciences nous apprend que, pendant plusieurs siècles, les hommes ont répugné à utiliser d'autres moyens que ceux du langage littéraire pour exprimer des concepts théoriques. Si un tel mode de communication, très adapté à l'homme, reste irremplaçable, et ces lignes en sont la preuve, la complexité des notions qui doivent être transmises impose un langage à la fois plus précis et moins abondant, tout en restant aussi compréhensible que le grec ancien ou le latin scientifique.

La présentation tabulaire de données ou de résultats fait partie de la vie courante : tableaux de pronostics de courses ou de présentation des chevaux et de leurs jockeys, tables des prix d'un magasin. L'idée de présenter dans une même table, non seulement des résultats mais aussi les conditions qui doivent être vérifiées pour que le résultat soit assuré est un phénomène récent. S. L. Pollack, dans un ouvrage que nous citerons dans ce chapitre, signale qu'en novembre 1957, GENERAL ELECTRIC lance un programme de recherche dont l'objectif était l'étude du flot des informations dans une usine. Quelle part un ordinateur peut-il prendre dans la détermination des actions à entreprendre de la commande à la livraison d'un client. La complexité de la logique d'un tel problème interdit l'usage des moyens d'expression habituellement usités : organigrammes, formules mathématiques ou langage courant. De là est venue la nécessité des "decision structure tables" dont la forme ressemblait à celle des tables de vérité dont elles étaient issues. On développa ensuite un outil pour programmer à partir de telles tables. Le processeur TABSOL a ainsi été implanté sur les séries 702 et 704 d'IBM, puis sur le GE 225 au début de 1961. Parallèlement, la documentation et l'analyse des algorithmes a été l'objet d'une étude, sans communication avec la précédente, réalisée par SUTHERLAND. Le résultat obtenu, bien que semblable quant aux concepts a une forme très différente, plus orientée vers la documentation.

L'étape suivante de l'emploi des tables de décisions est franchie en mai 1959. Le CONFERENCE on DATA SYSTEMS LANGUAGES (CODASYL) confie à l'un de ses comités, le SYSTEMS GROUP, le soin de développer un langage évolué indépendant de l'ordinateur. Il en résulte un langage de tables de décisions et un préprocesseur : DETAB-X (DEcision TABles eXpérimental).

L'histoire des tables de décisions au cours des années 1960 se confond avec celle des préprocesseurs de tables qui organisent la conversion des tables en programmes assimilables par la machine. La première génération formée de TABSOL, LOBOC, DETAB-X couvre les années 60-62 et se termine à l'occasion d'un séminaire tenu à New York les 20 et 21 septembre 1962 sous l'égide du CODASYL Systems group. Le résultat acquis au cours de cette période intéresse la forme des tables qui est encore celle utilisée aujourd'hui.

La seconde génération des processeurs de tables se caractérise par l'emploi de langage évolué, à la fois dans l'écriture du processeur lui-même, et comme langage cible du code produit. Les souches sont alors écrites en termes du langage utilisé. Il s'agit essentiellement de COBOL. En juin 1965, le Special Interest Group of Programming LAN gages (SIGPLAN) de l'Association for Computing Machinery (ACM) charge un groupe de travail d'étudier un tel préprocesseur : DETAB-65, implanté sur un grand nombre d'ordinateurs. Il faut signaler l'exception du Decision Logic Translator (DLT) produit par IBM pour la série 360 dont le langage cible est FORTRAN. On trouvera dans l'annexe bibliographique (18) une référence à tous les processeurs actuellement disponibles.

2.2 - DEFINITIONS GENERALES

Ce rapide rappel historique montre que les tables de décisions sont d'abord un moyen de communication entre l'homme et la machine, plus directement accessible à l'homme. La technique d'emploi des tables de décisions s'est accomodée de termes propres, dont les définitions, imposées par l'usage peuvent subir des modification d'un auteur à l'autre. La présentation que nous en faisons ici, s'appuie sur l'ouvrage référencé (2) dans l'annexe bibliographique :

DECISION TABLES. THEORY AND PRATICE.
Wiley and Sons Inc., 1971 New York.
par S. L. Pollack, H. T. Hicks et W. J. Harrison.

Nous n'en retiendrons ici qu'une présentation pratique des résultats. L'étude théorique contenue dans ce livre fait l'objet d'un résumé dans l'annexe I.

Après avoir défini les termes relatifs aux tables de décisions (2-2), nous décrirons plus précisément chacun des composants pour en déduire des propriétés des tables de décisions (2-3). Enfin nous présenterons les idées générales qui président à l'analyse d'un système à l'aide de tables de décisions (2-4).

2.2.1 - Structure de tables de décisions

Une règle de décisions peut s'exprimer par une phrase de langage courant, qui lie l'exécution d'une série d'actions à la valeur d'un ensemble de conditions. Par exemple :

> Si l'employé est payé à l'heure, et s'il a travaillé plus de 40 heures cette semaine, lui payer son salaire hebdomadaire habituel, augmenté du produit du taux horaire de l'heure supplémentaire par le nombre d'heures supplémentaires effectuées.

D'autres règles peuvent compléter celle-ci. Elles traduiront les actions à entreprendre dans le cas où l'employé horaire n'a pas fait d'heures supplémentaires, où l'employé est mensuel, etc. Toutes ces règles seront ensuite schématisées sur un tableau. Il aura la structure suivante :

TEXTE DES CONDITIONS	VALEURS DES CONDITIONS
TEXTE DES ACTIONS	EXECUTION DES ACTIONS

La partie gauche regroupe les textes. Nous dirons qu'elle contient les souches. Souches conditions et souches actions.

La partie droite est celle qui lie valeurs des conditions et exécution des actions. Elle contient les entrées des règles.

Chaque règle de décisions se lit dans une colonne de la partie entrées.

Notre exemple pourrait être traduit par la table suivante. On reconnaîtra la première règle de décisions. Le lecteur pourra aisément construire les phrases qui expriment les autres règles de la table.

SI	L'ouvrier est horaire	Ø	Ø	N	N
ET SI	L'ouvrier est mensuel	-	-	Ø	N
ET SI	L'ouvrier a travaillé plus de 40 heures	Ø	N	-	-
ALORS					
	Payer le salaire habituel	*	*		
ET	Ajouter le produit : taux * heures supplémentaires	*			
ET	Payer le salaire mensuel			*	
ET	Signaler une erreur				*

2.2.2 - Définitions des termes relatifs aux tables de décisions

Entrées limitées

On voit dans l'exemple des tables précédentes que les entrées conditions sont :

Ø si la condition doit être vérifiée pour que la règle soit satisfaite. (Y dans la littérature anglo-saxonne.)

N si la condition doit être fausse.

\- ou blanc, si la valeur de la condition n'importe pas pour la règle. La condition est sans objet pour cette règle. Nous dirons qu'il s'agit de l'entrée indifférente.

Les entrées actions sont notées :

* si l'action doit être entreprise dans la règle qui la contient.

blanc sinon.

Une table de décisions est dite à entrées limitées pour signifier que les seules entrées sont Ø, N, - pour la partie condition, * et blanc pour la partie action.

Entrées étendues

Les 4 règles de décisions de notre exemple peuvent encore être schématisées par le graphisme suivant :

L'employé est :	horaire	horaire	mensuel	autre
Il a travaillé plus de 40 heures :	Ø	N	-	-
Payer le salaire habituel	*	*		
Ajouter le produit : ...	*			
Payer le salaire mensuel			*	
Signaler une erreur				*

Nous y voyons des entrées limitées Ø, N, - et des entrées étendues. Dans le cas d'entrées étendues, une condition est exprimée par le rapprochement d'un texte de souche et d'une des entrées de la ligne.

Pour introduire la notion d'entrées actions étendues nous utiliserons l'exemple suivant :

Souche	Entrées des règles
ADD 10 TO A	*
ADD 10 TO B	*
ADD 10 TO C	*

Il peut être remplacé par :

Souche	Entrées des règles		
ADD 10 TO ...	A	B	C

Une table est dite à entrées étendues si entrées actions et conditions sont étendues. On dira d'une table qu'elle est à entrées mixtes si elle mélange les deux catégories d'entrées.

2.3 - PROPRIETES DES TABLES DE DECISIONS

2.3.1 - Conditions

2.3.1.1 - Postulats classiques

Les conditions qui permettent de séparer une règle de décisions sont classiquement liées par la conjonction ET. Ainsi soit la table :

X = 4	Ø	N
X = 6	-	Ø
U = 1	Ø	-
Z = 0	Ø	N
Q = 1	N	-
Z = 2	-	Ø

La première règle pourrait s'exprimer par la phrase COBOL :

```
IF  X = 4 AND  U = 1  AND  Z = 0  AND NOT  Q = 1  THEN ...
```

la seconde :

IF NOT X = 4 AND X = 6 AND NOT Z = 0 AND Z = 2 THEN ...

On voit dans ce dernier cas, que les valeurs de X et Z sont évaluées chacune deux fois de façon redondante : si X = 6 la condition NOT X = 4 est obligatoirement vérifiée.

Ainsi la notion classique de tables de décisions admet le postulat :

> La connaissance de la valeur d'une condition d'une règle ne permet pas de prévoir la valeur d'une autre condition de la règle.

L'énoncé suivant est équivalent : les conditions qui ont des entrées non indifférentes sur une règle sont traitées comme si elles étaient indépendantes.

Le second postulat classique traduit le fait que pour tout ensemble de valeurs données aux conditions de la table, une règle et une seule est satisfaite. La présence d'une règle AUTRE impose qu'au moins une règle soit satisfaite. La règle satisfaite doit être unique. C'est pourquoi la connaissance de chaque condition doit réduire le nombre de règles éligibles. Pour cela :

> Les entrées limitées d'une ligne condition ne peuvent être deux à deux que : identiques ou exclusives, ou l'une des deux être l'entrée indifférente.

Les entrées limitées classiques que nous avons choisies vérifient cette exclusion mutuelle. Dans le cas d'entrées étendues, ce postulat doit encore être vérifié. Deux entrées étendues d'une condition, dont aucune n'est l'entrée indifférente doivent ou s'exlure mutuellement ou être identiques.

*2.3.1.2 - Conditions dépendantes. Entrées * et §*

S. L. Pollack oppose à cette conception traditionnelle de la partie condition d'une table, une présentation que nous résumerons dans ce paragraphe.

Une condition est formée de deux opérandes supposés réduits chacun à une variable. L'un des deux opérandes peut être une constante. Les deux opérandes sont liés par un opérateur de comparaison classique :

	=	NE	GE	GT	LE	LT
pour :		$\neq$	$\geqslant$	$>$	$\leqslant$	$<$

Deux conditions C1 et C2, sont dépendantes si elles ont un opérande en commun. Le couple (C1, C2) ne peut alors prendre toutes les valeurs :

$$(\emptyset,\emptyset),\quad (\emptyset,N),\quad (N,\emptyset),\quad (N,N)$$

Si la valeur $(\emptyset,\emptyset)$ ne peut être atteinte les conditions sont dites exclusives. Par exemple :

C1 : J = 1 et C2 : J = 2

Dans les trois autres cas on parlera de dépendance avec recouvrement.

Entrées * et §

Ces entrées conditions seront utilisées pour indiquer une exclusion mutuelle entre deux conditions.

* sur la ligne d'une condition C_i signifie que si une autre des conditions de la règle vérifie l'entrée qui lui est demandée dans la règle (Ø ou N), la condition C_i est alors nécessairement fausse.

§ sur la ligne d'une condition C_j signifie que, si une autre des conditions de la règle vérifie l'entrée qui lui est demandée dans la règle (Ø ou N), la condition C_j est alors nécessairement vraie.

Ainsi soit la ligne d'entrées étendues :

VERBE =	'ADD"	'SUBTRACT'	'MULTIPLY'	'DIVIDE'

Elle sera traduite par la partie condition à entrées limitées suivante :

VERBE = 'ADD'	Ø	*	*	*
VERBE = 'SUBTRACT'	*	Ø	*	*
VERBE = 'MULTIPLY'	*	*	Ø	*
VERBE = 'DIVIDE'	*	*	*	Ø

On voit ainsi que dans cette représentation des tables de décisions, les deux postulats que nous avons posés sont refusés :

- les conditions qui interviennent dans une même règle peuvent être dépendantes,
- les entrées d'une même ligne peuvent ne pas être exclusives.

2.3.2 - Actions

Les actions d'une règle forment les étapes efficaces, celles qui réalisent quelque chose. Dans le cas de tables COBOL, une action est exprimée au moyen d'une instruction COBOL, d'une suite d'instructions, ou même d'une instruction conditionnelle :

GO TO P1 P2 P3 DEPENDING ON J.

Une même règle demande en général l'exécution de plusieurs actions. Celles-ci doivent être exécutées dans un ordre précis. Deux conventions sont généralement admises. Ou les actions sont écrites dans la souche dans l'ordre qui doit être le leur à l'exécution, ou l'ordre d'exécution est indiqué par un chiffre dans la partie entrée. Ces conventions s'appliquent aussi bien dans le cas d'entrées limitées que dans le cas d'entrées étendues. Ainsi la partie action suivante est correcte :

MOVE 1.15 TO Z	1	1) *	-	2	3) *
MOVE 2.4 TO Y	2	2) *	-	1	-
GO TO Q1	3	3) *	3) *	3	-
COMPUTE Z = ...	-	-	1) 1.15	-	1) 7.22 + Z
COMPUTE Y = ...	-	-	2) 2.4		2) Z * Y + 2
GO TO	-	-	-	-	4) Q2

Il arrive qu'une condition soit plus facilement évaluée au cours de l'exécution des actions. Son écriture dans la souche action est autorisée. Ainsi la phrase suivante est une ligne action correcte :

IF C1 THEN PERFORM P1 ELSE PERFORM P2	*	-	*

Si une telle ligne action contient n étoiles, l'écriture de la condition C1 dans la souche condition aurait eu pour conséquence la création de n règles supplémentaires.

Les différentes possibilités décrites par les exemples ci-dessus, ont été définies, dans le seul but de faciliter l'emploi des processeurs de tables. Certains préprocesseurs de tables autorisent encore que le contrôle puisse être passé des entrées actions d'une règle aux entrées actions d'une autre règle. D'autres offrent la possibilité d'exécuter des actions au cours de l'évaluation des conditions de la table. Les auteurs du livre déjà cité remarquent à ce propos, que le mécanisme d'une telle interprétation est difficile à imaginer, mais peut rendre de grands services. Nous aurons l'occasion au chapitre 3 de démonter ce mécanisme et d'en offrir une présentation simple, orientée vers la documentation, et non pas considéré comme une simplification d'écriture d'une table.

2.3.3 - Redondance et contradiction

Nous avons signalé que l'une des bases de l'interprétation des tables de décisions est que quelles que soient les valeurs des conditions, une seule règle est satisfaite. Nous étudions ici les conséquences de ce postulat.

Soit l'exemple de la table suivante :

Q = 1	Ø	N	N	Ø
P = 2	-	N	Ø	Ø
	R1	R2	R3	R4

On voit que les règles R1 et R2 de même que les règles R1 et R3 ou R3 et R4, s'exluent mutuellement. Suivant la valeur de Q l'une ou l'autre de ces règles est choisie. Les règles R2 et R3, R2 et R4, s'excluent mutuellement, elles sont distinguées par la valeur de la deuxième condition. Les règles R1 et R4 au contraire sont satisfaites simultanément si Q = 1 et P = 2. Nous dirons que la table présente une ambiguïté.

Si les actions des règles 1 et 4 sont identiques, on parlera de redondance, sinon, de contradiction.

Le caractère de non ambiguïté d'une table est lié à la présence d'entrées exclusives Ø et N pour séparer deux règles. Deux règles dont les entrées conditions diffèrent par au moins un couple Ø, N relatif à la même ligne condition ne peuvent être ambiguës. Soit l'exemple de la table ambiguë suivante :

C1	Ø	-	N
C2	-	Ø	N

Construisons la table de KARNAUGH équivalente à cette table :

	C1	$\overline{C1}$
C2	R1 / R2	R2
$\overline{C2}$	R1	R3

Une règle y est représenté par un pavé. Aux deux règles ambiguës de la table correspond une intersection non vide des pavés les représentant. L'intersection détermine les valeurs d'ambiguïtés des conditions.

On trouvera dans l'annexe bibliographique I plusieurs démonstrations du résultat suivant :

> Pour qu'une table de décisions à entrées limitées ne soit pas ambiguë, il faut et il suffit que les entrées conditions de 2 règles diffèrent par au moins un Ø et un N sur la même ligne.

Le résultat s'étend aux tables de décisions à entrées étendues, deux règles diffèrent par au moins un couple d'entrées exclusives.

2.3.4 - Table complète - Table réductible

Nous dirons qu'une table est complète si pour toute valeur des conditions une règle est satisfaite.

Une table qui contient une règle Autre est complète. Nous discutons ici les tables à entrées limitées sans règle Autre.

Une règle est dite simple si aucune de ses entrées conditions n'est l'entrée indifférente. Elle est représentée sur la table de KARNAUGH par un pavé "élémentaire". Une table à n conditions, dont toutes les règles sont simples devra avoir 2^n règles pour être complète.

Les deux tables suivantes sont équivalentes :

C1	Ø	Ø	Ø	Ø	N	N	N	N
C2	Ø	Ø	N	N	Ø	Ø	N	N
C3	Ø	N	Ø	N	Ø	N	Ø	N
GO TO	P1	P1	P1	P1	P2	P2	P3	Q

C1	Ø	N	N	N
C2	-	Ø	N	N
C3	-	-	Ø	N
GO TO	P1	P2	P3	Q

La première est complète. Considérons-en les règles 5 et 6. L'action entreprise est la même quelle que soit la valeur de C3. Ces deux règles peuvent être résumées par la seule règle 2 de la deuxième table.

Nous dirons que la première table est réductible. Par application des mêmes remarques, les règles 1 à 4 de cette table peuvent être réduites pour former la règle 1 de la dernière table.

La table 2 est donc complète. Une règle qui contient r entrées indifférentes est équivalente à 2^r règles simples. Nous appellerons 2^r le poids de la règle, il mesure la "surface" occupée par la règle sur la table de KARNAUGH. Une table à n conditions est complète si la somme des poids de ses règles vaut 2^n. On trouvera, encore, plusieurs démonstrations de ce résultat dans l'annexe I.

2.4 - PRATIQUE DES TABLES DE DECISIONS - ANALYSE DES SYSTEMES

La forme des tables de décisions que nous avons envisagée est entièrement orientée vers l'utilisation des préprocesseurs de tables. Il s'agit de programmes qui assurent la traduction automatique des tables en programmes de langage évolué (COBOL, FORTRAN). La littérature qui concerne les algorithmes de traduction est très abondante. On en trouvera plusieurs références dans l'annexe bibliographique.

L'informaticien conserve la responsabilité de l'écriture des tables et de l'analyse détaillée du système qu'il envisage de créer. L'utilisation des tables de décisions est orientée vers la description des programmes dits de gestion. Pour ces programmes nous distinguerons la phase d'analyse du problème (System Analysis) de la phase construction de l'architecture du programme (Systems design).

Au cours de la première, l'analyste prend connaissance des différents cas à traiter, pour préparer le travail d'étude du programme qui traitera ces cas. Dans la seconde phase il organise pratiquement le traitement.

La première phase peut être décomposée en plusieurs étapes :

1) Identifier et délimiter le problème. Par exemple : paie des employés d'une entreprise.

2) Décomposer le problème en facteurs relativement indépendants. Il leur correspondra des modules dans la deuxième phase. Traitements bruts, charges patronales, Sécurité Sociale, Calcul des montants imposables, Edition.

3) Rassembler la documentation utile. En particulier sur la forme des données à traiter.

4) Identifier en les détaillant toutes les données qui devront intervenir : fichiers d'entrées et de sorties.

La construction des tables de décisions s'accompagne de vérifications constantes sur la validité des tables. L'analyste doit se poser les questions suivantes :

1) Quel est le domaine des valeurs possibles pour chaque variable qui intervient dans les conditions.

2) Est-ce que les données d'entrées ont bien la signification qui leur est donnée dans les conditions de la table. Par exemple telle variable indique-t-elle les seules heures supplémentaires ou les heures totales, dont il faudra soustraire le nombre d'heures de base.

3) Est-ce que toutes les conditions qui interviennent dans le problème sont présentes dans la souche condition.

4) La table est-elle complète.

5) Contient-elle des ambiguïtés qui pourraient correspondre à des erreurs.

2.5 - CONCLUSION

Les tables de décisions ont fait l'objet d'une présentation intuitive dans ce chapitre. Elles y apparaissent comme un moyen de communication privilégié entre l'homme et la machine. Les chapitres qui suivent chercheront à définir les tables de décisions comme un moyen de communication entre la machine et l'homme.

CHAPITRE 3

DÉFINITIONS RELATIVES AUX TABLES DE DÉCISIONS A ENTRÉES LIMITÉES

Les tables de décisions ont été créées pour permettre une communication de l'homme vers la machine : générations de programmes de traitement à partir de tables d'analyse. Le point de départ humain a imposé les définitions, consacrées ensuite par l'usage, des termes d'utilisation : règles, souches, entrées. Dans le chapitre suivant nous aurons à construire la démarche inverse :

A partir d'un programme source, éditer des tables de documentation. Pour atteindre ce but nous proposons ici une définition des tables de décisions que nous emploierons et des termes qui leur sont relatifs. Nous attacherons ces définitions aux notions de programme et de graphe connexe.

3.1 - POSITION DU PROBLEME - NOTATIONS

Après avoir défini quelques mots de vocabulaire propres à la théorie des graphes, nous établirons une définition théorique d'un programme. Nous pourrons alors poser la définition d'une table de décisions à entrées limitées, définition qu'il sera ensuite possible de comparer, pour la justifier, à la notion classique de tables de décisions telle qu'elle a été rappelée au chapitre précédent.

Les définitions retenues seront exploitées dans les chapitres suivants.

3.1.1 - Rappels et définitions

Un graphe G est un couple constitué d'un ensemble fini X et d'une par-

tie U du produit cartésien X ✱ X :

$$G = (X , U)$$

Les éléments de X sont appelés les sommets du graphe.

Les éléments de U sont les arcs. Le graphe est orienté si l'existence dans U d'un arc (x_1 , x_2) n'entraîne pas l'existence de l'arc symétrique : (x_2 , x_1).

Si (x_1 , x_2) est un arc nous dirons que x_1 est son extrémité initiale et x_2 son extrémité finale.

Nous appellerons degré intérieur (certains auteurs parlent de demi-degré) d'un sommet du graphe le nombre d'arcs d'extrémités finales ce sommet, et degré extérieur d'un sommet le nombre d'arcs admettant le sommet comme extrémité initiale.

Nous appellerons sous-graphe d'un graphe G = (X,U) un graphe : G' = (X',U') tel que X' soit inclus dans X et U' la restriction à X' des arcs de U, c'est-à-dire l'intersection des ensembles U et X' ✱ X'.

Un chemin d'un graphe est une suite d'arcs telle que l'extrémité finale de tout arc de la suite, autre que le dernier, coïncide avec l'extrémité initiale de l'arc suivant. Le chemin est généralement décrit par la suite des sommets qu'il rencontre. Si cette suite de sommets ne contient pas de répétition le chemin est dit élémentaire.

La longueur d'un chemin pourra être ici le nombre d'arcs qui le décrit.

Un circuit est un chemin fini dans lequel le sommet initial coïncide avec le sommet terminal.

On appelle base, ou noyau, d'un graphe un sous-ensemble D, s'il existe, de sommets tels que :

- si x et y sont des sommets de la base il n'existe pas de chemin (x,y)
- et que pour tout sommet y hors de la base il existe un chemin d'extrémité finale y dont l'extrémité initiale est dans la base.

Une racine est une base réduite à un seul sommet.

Une arborescence est un graphe ayant une racine x_0 et tel que tout sommet autre que la racine a un seul précédent. L'existence de la racine impose qu'il en est au moins un. Une arborescence ne possède pas de circuit.

Un graphe est fortement connexe si deux sommets quelconques x et y sont reliés par au moins un chemin. On montre alors que la relation binaire construite sur l'ensemble des sommets :

xRy s'il existe un circuit passant par x et y ou x et y sont confondus

est d'équivalence. On appelle composante fortement connexe relative à un sommet x_0 la classe d'équivalence C_0 à laquelle appartient x_0.

Le graphe réduit G_r d'un graphe $G = (X,U)$ est obtenu en prenant :

- pour ensemble de sommets l'ensemble quotient de x par la relation d'équivalence,

- pour ensemble d'arcs un ensemble U' tel que :

$(C_i, C_j) \in U' \leftrightarrow \exists x_i \in C_i$ et $x_j \in C_j$ tels que $(x_i, x_j) \in U$.

On notera qu'un graphe réduit est sans circuit.

3.1.2 - Programme abstrait

Pour attacher la définition d'une table de décisions à la notion de programme, nous recherchons en premier lieu une définition théorique d'un programme ou programme abstrait.

Nous disposons des ensembles suivants définis de façon classique :

- A est l'ensemble des assignations. Elles sont de la forme :

$$L \leftarrow b \ (L_1, \ \ldots \ , L_m)$$

où b est un symbole d'opérations portant sur m arguments, ou locations ou variables, du programme. Le résultat de l'opération est affecté à l'argument L.

- C est l'ensemble des prédications de la forme :

$$c = p \ (L_1, \ \ldots \ , L_n)$$

où p est un symbole de prédicat. Il peut prendre deux valeurs $\emptyset$ (ou Y) ou N. $L_1 \ \ldots \ L_n$ sont encore des arguments du programme.

Un programme abstrait est alors défini comme un graphe orienté, connexe fini. L'ensemble X générique de ses sommets est la réunion logique de sous-ensembles finis de C et de A. Nous dirons par la suite, que chaque sommet est étiqueté par une prédication ou une assignation, et par abus de langage que le sommet est une prédication ou une assignation.

Deux sommets, singularisés plus bas, portent, l'un l'étiquette E (pour entrée) l'autre l'étiquette S (pour sortie). A chaque arc du graphe est attachée une étiquette : Ø, N ou ✱.

Par définition, le graphe définissant un programme abstrait jouit des propriétés suivantes :

- Il existe un seul sommet de degré intérieur nul. Ce sommet est le seul à porter l'étiquette spéciale E. Il est de degré extérieur 1.

- Il existe un seul sommet de degré extérieur nul. Ce sommet est le seul à porter l'étiquette spéciale S.

- Chaque sommet autre que l'entrée ou la sortie :
  - ou bien est de degré extérieur 2, il est alors étiqueté par une prédication et les deux arcs sont étiquetés l'un par Ø, l'autre par N ;
  - ou bien est de degré extérieur 1, le sommet est alors étiqueté par une assignation et cet arc est étiqueté par le symbole ✱.

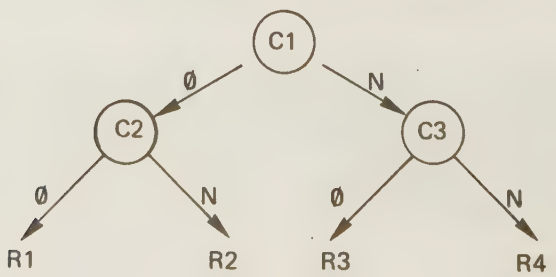
Cette définition du programme abstrait posée, nous rechercherons à singulariser les sous-graphes de ce programme, qui sont susceptibles d'être représentés par une table de décisions.

3.1.3 - Titre d'un sous-graphe

La traduction de la partie condition d'une table en un graphe, telle qu'elle est organisée dans la démarche classique de l'utilisation des tables des décisions, conduit à une arborescence binaire. Plusieurs algorithmes de traduction sont étudiés dans l'annexe bibliographique. Ainsi la table P1 suivante peut être traduite par l'organigramme indiqué :

Table P1

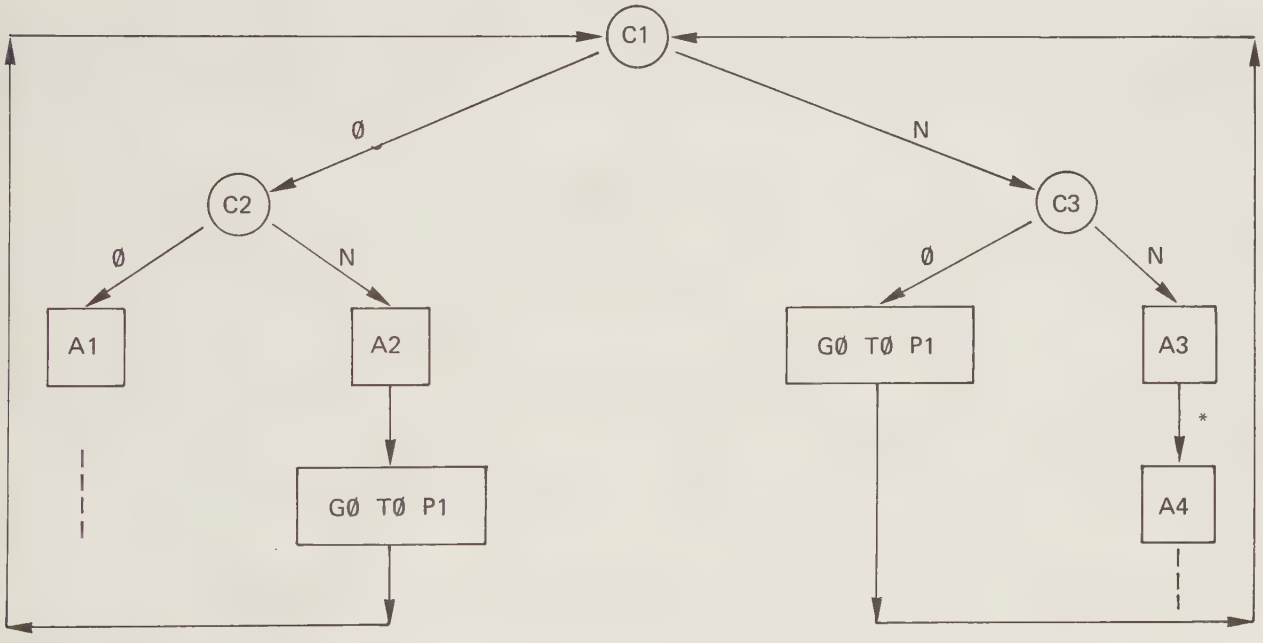
	R1	R2	R3	R4
C1	Ø	Ø	N	N
C2	Ø	N		
C3			Ø	N



Le graphe ainsi obtenu n'est pas encore un graphe de programme. Aucun sommet n'est étiqueté par une assignation. Et nous ne pouvons conclure qu'à une table de décisions complète correspond un graphe à structure d'arborescence binaire. Si l'on complète la table par la partie action suivante dans laquelle une action a été précisée.

	R1	R2	R3	R4
A1	*			
A2		*		
A3				*
A4				*
GØ TØ P1		*	*	

Le graphe complet devient :



L'arborescence précédente est disparue. Le graphe obtenu contient des circuits. Ils correspondent aux ruptures de séquences vers elle-même que contient la table de décisions. La racine de l'arborescence primitive appartient à tous les circuits du graphe complet.

Ainsi le graphe qui traduit une table de décisions obéit à l'une ou l'autre des structures schématisées aux figures 1 - 2 - 3 ou 4. Les trois premières sont des arborescences dont les racines sont E1, E2, E3. La quatrième comporte des circuits.

Fig. 1

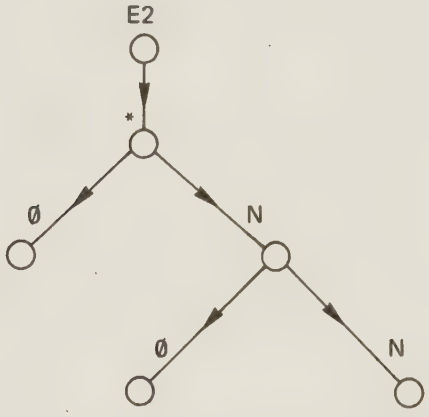


Fig. 2

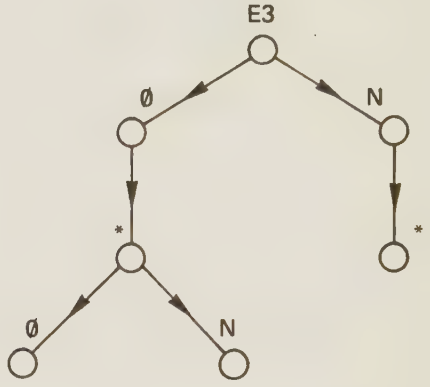


Fig. 3

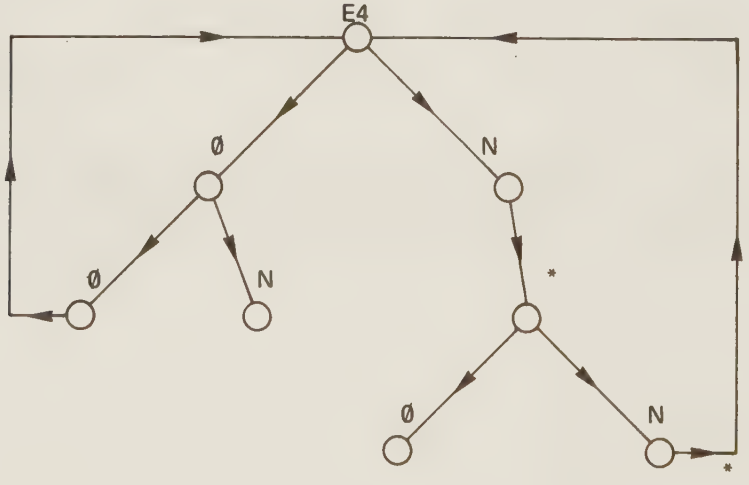
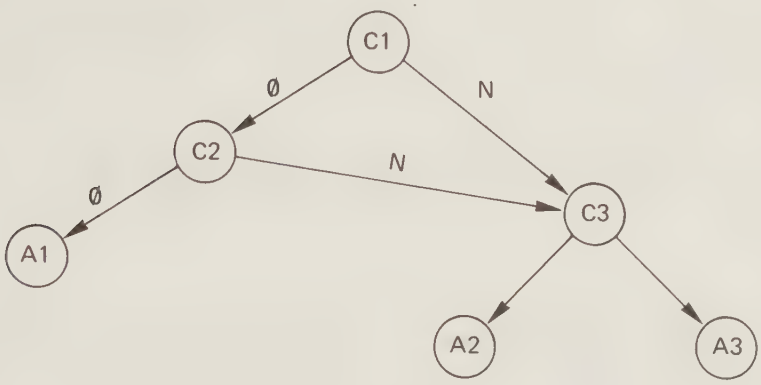


Fig. 4

L'interprétation classique d'une table de décisions interdit d'évaluer deux fois une même condition ou la même action le long d'une règle. Ceci se traduit par le fait que tous les sommets des graphes ci-dessus, sauf E4, sont de

degré intérieur 1. Nous ne ferons correspondre, du moins directement, aucune table à un graphe tel que le suivant, qui n'est pas une arborescence :



Les circuits du schéma 4 obligent à distinguer la notion de titre de sous-graphe, de la notion de racine d'arborescence ; nous sommes amenés à poser la définition suivante qui permet de particulariser les sous-graphes représentables par une table de décisions.

Définition

Un sous-graphe de programme admet un titre, s'il existe un sommet T, le titre, qui satisfait aux conditions suivantes :

a) T appartient à tous les circuits du sous-graphe.

b) Tous les sommets sauf éventuellement le titre sont de degré intérieur 1.

c) Pour tout sommet X du sous-graphe autre que T il existe un chemin élémentaire (T, X).

La connexité du graphe d'un programme entraîne que si le titre existe, il est unique.

S'il existe un titre le chemin (T, X) est unique. La démonstration résulte du fait que le précédent de X sur ce chemin est unique d'après *b)*, et ainsi de proche en proche l'unicité du chemin.

Si le graphe n'a pas de circuits et s'il admet un titre, ce titre est la racine du graphe et le graphe est une arborescence. Cette proposition, déduite

immédiatement de la définition, conduit à considérer la notion de titre comme une extension de la notion de racine d'arborescences à des graphes comportant des circuits.

Les sommets E1, E2, E3, E4 des 4 figures précédentes sont les titres des graphes correspondants.

3.2 - DEFINITION DES TABLES DE DECISIONS A ENTREES LIMITEES

Nous avons signalé, au paragraphe précédent, et sans aucune démonstration, les structures de graphes susceptibles de représenter une table de décisions telle qu'elle est utilisée classiquement. Nous en avons dégagé la notion de titre de sous-graphe. Nous attacherons maintenant la définition d'une table de décisions aux sous-graphes qui ont été particularisés.

Définition

Une table de décisions à entrées limitées est un sous-graphe d'un programme abstrait admettant un titre.

Cette définition théorique, introduite par les paragraphes précédents sera justifiée, dans les chapitres 4 et 5, par la richesse des conséquences qu'elle entraîne.

La définition posée pourrait déjà permettre de traduire un programme en un ensemble de tables de décisions à entrées limitées. La recherche de tous les sous-graphes du graphe du programme qui admettent un titre, c'est-à-dire qui ont une structure de tables de décisions conduirait à un grand nombre de tables de très petites "dimensions". Ainsi aux phrases suivantes :

```
        IF   C1   THEN   A1   ELSE   A2.
        A3.
P2.
        A4.
        GO TO P2.
```

devraient correspondre les trois tables de décisions :

Table 1

C1	Ø	N
A1	*	
A2		*
Table suivante	*	*

Table 2

A3	*
GO TO P2	*

Table P2

A4	*
GO TO P2	*

La séparation en deux tables T1 et T2 est liée au fait que le sommet A3 a deux antécédents. La table P2, qui contient un circuit, doit être séparée.

A l'évidence une telle solution n'est pas envisageable. Nous décrirons au chapitre 4 les transformations préalables que doit subir le graphe équivalent au programme. Celles-ci permettront de lever une telle difficulté. Nous y détaillerons également l'algorithme qui permet d'éditer sous la forme classique d'un tableau, les tables de décisions définies ici à l'aide d'un graphe.

3.3 - TERMES USUELS RELATIFS AUX TABLES DE DECISIONS

3.3.1 - Aïeul - Ruptures de séquences

- Nous appellerons aïeul A d'un sommet S d'une table de décisions un sommet tel qu'il existe un chemin (A, S) ne passant pas par le titre, et par conséquent, élémentaire.

- Nous appellerons rupture de séquence l'étiquette portée par un sommet d'une table de décisions, n'ayant pas de suivants ou dont le seul suivant est le titre de la table.

L'étiquette GO TO P2 de l'exemple précédent est une rupture de séquence. La présence de circuits impose cette distinction de la notion d'ancêtre (pour aïeul) et de sommet pendant (pour rupture de séquence), classiquement défini en théorie des graphes.

3.3.2 - Règles - Souches - Entrées

Après avoir défini une table de décisions à l'aide d'un graphe, il est possible de reconnaître sur le graphe les éléments classiques d'une table de décisions.

Nous appellerons règle un chemin élémentaire de la table de décisions, dont l'extrémité initiale est le titre de la table et l'extrémité finale une rupture de séquence.

Il y a dans cette acception du terme autant de règles que de ruptures de séquence, c'est-à-dire de branches distinctes dans l'arborescence générique. Nous examinerons au paragraphe 3.4 suivant, les conclusions déduites de cette remarque.

Le graphe étant fini, le nombre de ruptures de séquence est fini, l'est aussi le nombre N de règles. A chaque règle est affecté un numéro d'ordre i varian de 1 à N.

Munissons l'ensemble S_{ai} des étiquettes portées par les sommets de la règle i et qui ont pour degré extérieur 1, de la relation binaire naturelle :

A R_i B si le sommet étiqueté par A est un aïeul du sommet étiqueté par

Cette relation binaire est transitive, c'est une relation d'ordre strict e total sur S_{ai}.

L'ensemble S_a union logique des ensembles S_{ai} :

$$S_a = \bigcup_{i=1}^{N} S_{ai}$$

permet de construire un graphe G_a dont chaque sommet est un élément de S_a, c'est-à-dire une assignation. Un arc (A, B) est créé si une relation A R_i B est vérifiée

L'application de la règle suivante à tous les couples de sommets identique conduit à un nouveau graphe G_a^*.

Règle

Pour deux sommets identiques A_1 et A_2 de G_a tels qu'il n'existe pas de chemin dans G_a liant A_1 à A_2 les arcs (A_1, A_2) et (A_2, A_1) sont créés. Le graphe G_a est ensuite remplacé par son graphe réduit.

Le graphe G_a n'ayant pas de circuit, la transformation décrite supprime les sommets identiques A_1 et A_2 pour les remplacer par un sommet unique. Le graphe G_a^* obtenu par application de cette règle à tous les couples d'assignations identiques, n'a pas de circuit puisqu'obtenu par réduction à ses composantes fortement connexes d'un graphe fini. Définissons sur l'ensemble S_a^* des sommets de G_a^* la relation binaire R suivante :

A R B s'il existe un chemin (A, B) dans G_a^*

Cette relation est transitive, c'est donc une relation d'ordre. Le graphe G_a^* n'étant pas connexe, c'est une relation d'ordre partiel.

Par définition, nous appellerons souche actions l'ensemble S_a^* muni de cette relation d'ordre partiel.

La souche conditions est par définition l'ensemble des étiquettes portées par les sommets de degré extérieur 2 du graphe de la table de décisions. Si cet ensemble est l'ensemble vide, la table est dite inconditionnelle.

Nous appellerons entrées d'une règle la suite ordonnée des étiquettes portées par les arcs décrits le long de la règle. Chaque étiquette est associée au sommet origine de l'arc qui la porte. Il faut distinguer les entrées actions (*), le sommet origine de l'arc est une assignation, des entrées conditions (Ø, N) le sommet origine de l'arc est une prédication.

3.3.3 - Propriétés

Avant de discuter l'intérêt et le bien-fondé des définitions que nous avons posées, signalons deux propriétés essentielles des tables de décisions telles qu'elles ont été définies.

3.3.3.1 - Table de décisions complète

Les tables de décisions que nous avons définies sont complètes, c'est-à-dire que quelles que soient les valeurs affectées aux conditions, une règle est satisfaite. Cette propriété est immédiatement déduite de la définition du

programme abstrait, qui impose que les sommets étiquetés par un prédicat aient deux suivants.

Le paragraphe suivant permettra de conclure que la règle vérifiée est unique.

3.3.3.2 - Ambiguïté - Contradiction

Rappelons que les entrées conditions d'une table de décisions sont dites ambiguës, si pour un ensemble de valeurs données aux conditions de la table, plusieurs règles sont satisfaites. Si les entrées actions de ces règles sont identiques la table est dite ambiguë, sinon les règles sont dites contradictoires.

Il est possible de reconnaître simplement le caractère ambigu des entrées conditions d'une table de décisions sans s'astreindre à l'essai exhaustif de toutes les combinaisons possibles des valeurs des prédications. Nous avons signalé au chapitre 2 qu'une table de décisions à entrées limitées n'est pas ambiguë si et seulement si les entrées conditions de deux règles quelconques diffèrent par au moins un Ø et un N affectés au même prédicat.

Les tables de décisions telles que nous les avons définies ne sont ni ambiguës ni contradictoires.

Pour montrer ce résultat nous chercherons à vérifier la condition nécessaire et suffisante qui vient d'être rappelée. Deux règles différentes, c'est-à-dire deux chemins différents dans le graphe de la table, ont une partie commune, réduite au titre éventuellement. Seuls les sommets étiquetés par un prédicat ont deux suivants. Le dernier élément de la partie commune de deux règles est donc un tel sommet. Les entrées conditions des deux règles diffèrent par l'entrée affectée à ce prédicat : Ø pour l'une, N pour l'autre. Les deux règles ne sont donc ni ambiguës ni contradictoires et de même la table entière.

3.4 - TABLES DE DECISIONS CLASSIQUES - DISCUSSION DE LA DEFINITION

Nous avons posé une définition des tables de décisions sensiblement différente des définitions habituellement utilisées. Les deux questions réciproques suivantes se posent pour conclure à la validité de notre définition :

3.4.1 - Les tables de décisions définies dans ce chapitre correspondent-elles à la notion classique de tables de décisions ?

3.4.2 - Une table de décisions obéit-elle dans tous les cas à la définition proposée ici ?

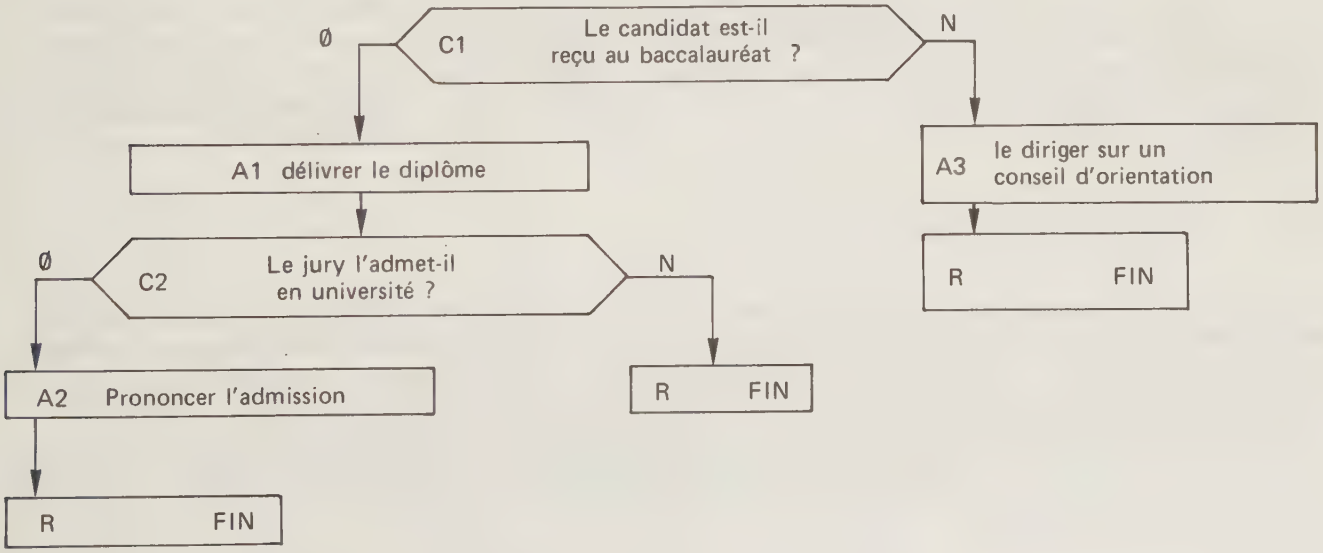
Nous discuterons la réponse qu'il convient de leur apporter. Les tables de décisions telles qu'elles sont utilisées, n'ont fait l'objet, à ce jour, d'aucune définition théorique, du moins à notre connaissance. Il s'agit pour les utilisateurs, d'un graphisme dont l'emploi se révèle efficace lors de l'analyse et de la documentation d'un algorithme. Les tables de décisions en offrent une représentation claire et synthétique. Dans la mesure où ces qualités seront conservées, voire accrues, nous nous permettrons de proposer quelques extensions à la forme classique d'édition des tables de décisions.

3.4.1 - Dans la définition apportée d'une table de décisions il n'a été fait aucune hypothèse sur l'antécédence relative des sommets étiquetés par une condition et ceux qui sont étiquetés par une assignation. Ainsi :

- Le titre d'une table de décisions tel que défini, peut être étiqueté par une assignation.

- Un sommet étiqueté par une assignation peut avoir un suivant étiqueté par une condition.

Au contraire, l'utilisation classique d'une table de décisions impose que toutes les conditions soient évaluées, et la règle satisfaite déterminée avant que la première action puisse être entreprise. A une telle organisation de l'interprétation de la table correspond un organigramme dans lequel un sommet de degré extérieur 1 ne peut avoir pour suivant un sommet de degré extérieur 2.



C1	Le candidat est-il reçu au baccalauréat	Ø	Ø	N
C2	Le jury l'admet-il en université	Ø	N	
A1	Délivrer le diplôme	*	*	
A2	Prononcer l'admission	*		
A3	Le diriger sur conseil d'orientation			*
R	FIN	*	*	*

Ainsi la table de décisions classique et le graphe à structure de table de décisions dessinés ci-dessus ne procèdent pas de la même utilisation. Il reste évident qu'ils organisent le même algorithme et que les résultats produits à l'aide de l'un ou de l'autre seront dans tous les cas identiques. Il faut remarquer que l'exécution de l'action A1 n'intervient pas dans l'évaluation de la condition C2.

3.4.1.1 - Ruptures sémantiques

Nous nous placerons maintenant dans le cas contraire.

Si A est un argument, une assignation telle que :

$$A \leftarrow b\ (A1, \ldots, An)$$

ne peut être, dans l'acception classique des tables de décisions, l'aïeul d'un sommet condition dont un argument est A : p $(A, A'_1, \ldots, A'_m)$.

Nous appellerons rupture sémantique, ou rupture de table liée à la sémantique, la présence d'un tel événement. La présence d'une rupture sémantique dans le graphe d'une table de décisions interdit l'édition sous forme classique de la table. Trois solutions peuvent être envisagées. Nous les étudierons l'une après l'autre.

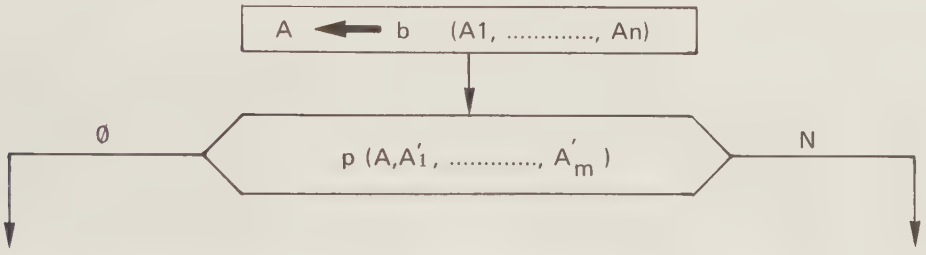
3.4.1.2 - Edition séparée

La première est d'apporter une restriction à la définition des tables de décisions de façon à interdire la présence de ruptures sémantiques. Nous y voyons deux inconvénients. Les tables de décisions ont été définies par référence à la structure de sous-graphes de programme abstrait, une restriction liée à la sémantique de ces sous-graphes pourrait paraître hétérogène. D'autre part une telle limitation, quelle qu'elle soit imposerait l'édition de plusieurs tables classiques.

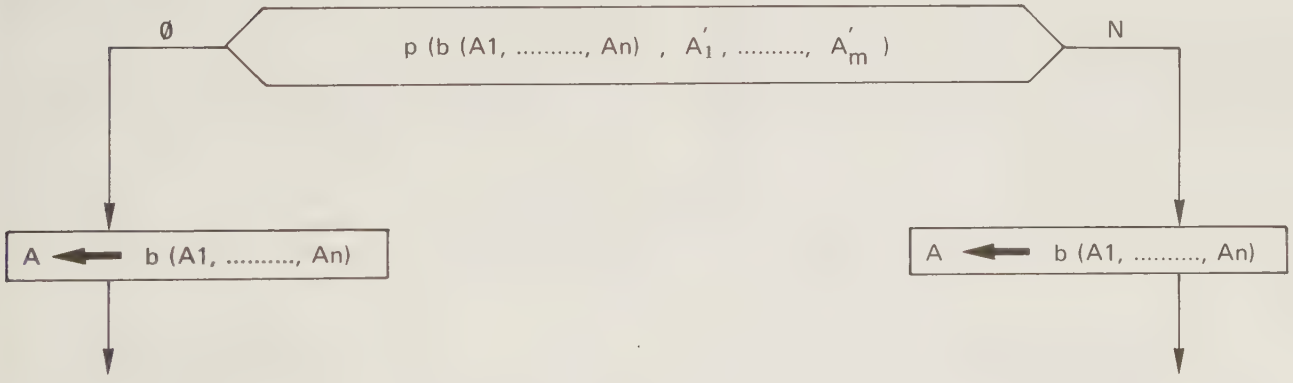
Le caractère synthétique de la représentation par tables de décisions risque alors d'être perdu. L'expérience montre que le phénomène de rupture sémantique est fréquent dans tous programmes réels. Nous conserverons donc la définition d'une table de décisions proposée dans la première partie de ce chapitre.

3.4.1.3 - Substitution

Une rupture sémantique est caractérisée par la présence d'une location A, réceptrice dans une assignation et argument d'une condition suivant le schéma de la figure :



Nous pouvons envisager de construire le schéma sémantiquement équivalent suivant, obtenu par substitution, dans la condition, de la location A, par



l'expression qui lui est affectée.

La rupture sémantique a ainsi disparu. Cette transformation par substitution présente l'inconvénient de conduire à un codage lourd et surtout à une documentation moins claire. Nous abandonnerons cette solution, dans la mesure où il faut obtenir des tables de décisions qui, éditées, restent compréhensibles. Cependant cette transformation servira de point de départ à l'évaluation du programme. Nous en étudierons les conséquences au chapitre 6.

3.4.1.4 - Sous-tables de décisions

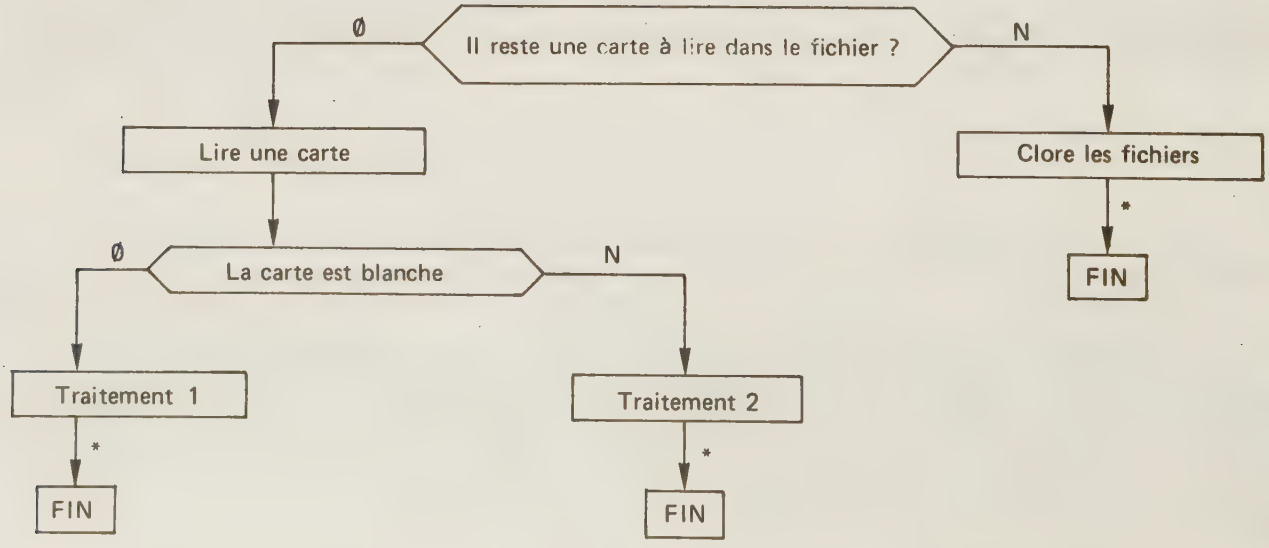
Par référence à la définition d'une table de décisions nous appellerons sous-table de décisions, un sous-graphe de la table admettant un titre, c'est-à-dire à structure de table de décisions, et dont les ruptures de séquence sont des ruptures de séquences de la table entière.

La dernière partie de la définition impose que la sous-table est entièrement définie par la donnée de son titre qui peut être l'un quelconque des sommets de la table entière, action ou condition. Les derniers sommets de chaque règle, ou rupture de séquence, sont aussi les derniers sommets des règles de la table principale.

Deux sous-tables de décisions seront dites imbriquées si l'une d'elle est une sous-table de l'autre. La relation binaire d'imbrication est transitive. Elle induit donc une relation d'ordre partiel dans l'ensemble des sous-tables d'une même table de décisions. Deux sous-tables de décisions pour lesquelles la relation d'imbrication n'est pas vérifiée seront dites parallèles. Les titres de deux sous-tables parallèles appartiennent à deux branches différentes de l'arborescence générique de la table.

Les notions de sous-tables imbriquées et de sous-tables parallèles seront exploitées au chapitre 5 lors de la construction des tables de décisions à entrées étendues.

Sur la forme éditée, la sous-table de décisions pourra être particularisée de façon à séparer souches conditions et actions, suivant qu'une condition ou une action appartient ou non à la sous-table. Ainsi soit l'exemple suivant : le graphe indiqué est celui d'une table présentant une rupture sémantique. Le prédicat qui détermine la rupture est le titre de la sous-table qui a été singularisée.



La table de décisions éditée peut alors être interprétée de façon à respecter la rupture sémantique.

Il reste une carte dans le fichier	Ø	Ø	N
Sous-table 1			
La carte est blanche	Ø	N	
Lire une carte	*	*	
Clore les fichiers			*
Sous-table 1			
Traitement 1	*		
Traitement 2		*	
Ruptures de séquence			
FIN	*	*	*

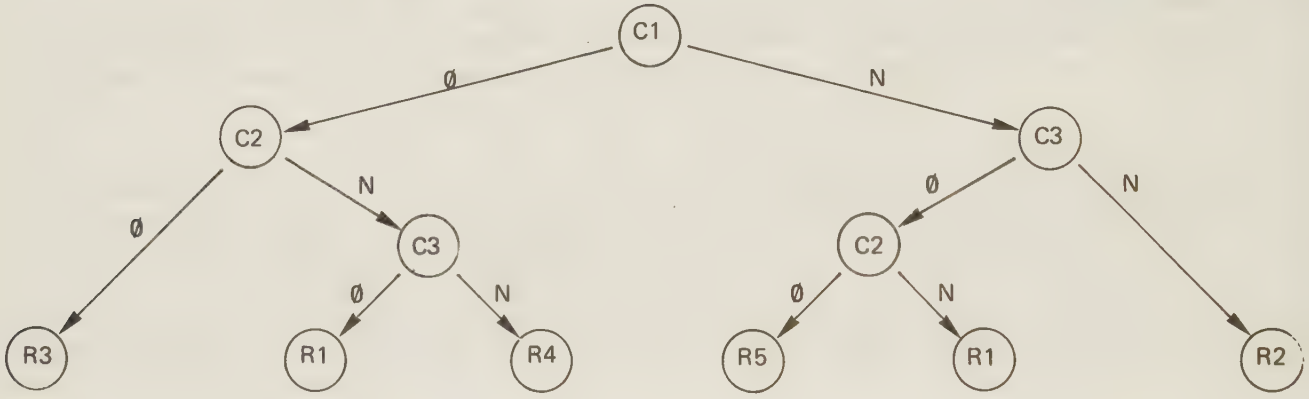
Les premières conditions sont évaluées, pour chaque règle les premières actions sont entreprises s'il en existe, ensuite la dernière partie de la table est interprétée de façon classique. Enfin les ruptures de séquence qui terminent chaque règle sont exécutées. Le tableau ainsi interprété reste lisible, il garde en outre l'intérêt d'offrir une représentation synthétique. Nous adopterons cette solution pour représenter une rupture sémantique.

De façon générale, des sous-tables imbriquées seront éditées en séparant les souches autant que nécessaire, et dans l'ordre induit par la relation d'imbrication. Les sous-tables parallèles sont éditées dans un ordre arbitraire. Remarquons qu'en présence de sous-tables imbriquées, les sous-tables supérieures présentent des colonnes identiques, les entrées conditions et actions de la sous-table sont identiques dans plusieurs colonnes. Les règles sont différenciées par les colonnes suivantes.

3.4.2 - Les tables de décisions que nous avons définies à l'aide de graphes ne couvrent pas l'ensemble des tables de décisions classiquement utilisées. Il nous suffira d'en donner un exemple :

C1	-	N	Ø	Ø	N
C2	N	-	Ø	N	Ø
C3	Ø	N	-	N	Ø
	R1	R2	R3	R4	R5

Cette table de décisions est complète et sans ambiguïtés. Sa particularité est la suivante. Un organigramme équivalent à cette table aura au moins deux branches pendantes qui correspondront à la même règle. Ainsi si C1 est la première condition de l'organigramme, les actions, non représentées ici, de la règle 1 devront être programmées sur deux sommets pendants de l'arborescence construite. L'un sera sur la branche descendant de l'arc Ø de C1, l'autre de l'arc N. Il en sera de même pour la règle2(respectivement 3). si c'est la condition 2 (resp. 3) qui est programmée la première. La table de décisions étudiée n'admet pas de titre.



Les algorithmes qui transforment une table de décisions en un organigramme ne s'intéressent pas à la partie action de la table. Aucune étude ne s'est inquiétée de l'inconvénient que présente une telle table. L'organigramme qui la traduit et que nous avons dessiné, a une structure de table de décisions. Cependant la forme éditée correspondante est réductible. Notre exemple conduirait à l'édition du tableau :

C1	Ø	N	N	Ø	Ø	N
C2	N	N	-	Ø	N	Ø
C3	Ø	Ø	N	-	N	Ø
	R1	R1	R2	R3	R4	R5

qui n'est pas celui de départ.

Par définition une table de décisions à entrées limitées est réductible s'il existe deux règles dont les entrées (actions et conditions) ne diffèrent que par un couple Ø, N relatif à la même ligne condition. La table peut alors être transformée en une table contenant une règle de moins et sémantiquement équivalente.

Les tables de décisions telles que nous les avons définies peuvent être réductibles.

L'exemple que nous avons détaillé montre que les tables de décisions permettent de représenter des structures logiques non représentables par organigrammes arborescents. Nous montrerons au chapitre suivant qu'une telle table irréductible ne peut être codée par une phrase COBOL du type :

IF ... THEN ... ELSE ...

Nous dirons que la table est non-programmable.

3.4.3 - Conclusion

Il nous faut donc conclure que les tables de décisions que nous avons définies dans ce chapitre ne correspondent pas exactement à la notion classique de table de décisions. Le chapitre suivant justifie la définition que nous avons posée en montrant qu'elle permet la traduction d'un programme donné, écrit en COBOL, en un ensemble de tables de décisions.

CHAPITRE 4

CONSTRUCTION DES TABLES DE DÉCISIONS A ENTRÉES LIMITÉES D'UN PROGRAMME COBOL

4.1 - PRESENTATION - TABLE DE DECISIONS ABSTRAITE

La définition des tables de décisions à entrées limitées proposée au chapitre précédent, utilise essentiellement la notion de graphe. La traduction d'un programme en un jeu de tables de décisions pourrait donc se résumer en la recherche d'arborescences ou de sous-graphes particuliers, dans le graphe complet du programme. Cette démarche présente l'inconvénient de détailler la structure du programme donné, et elle seule, alors que chaque table, sous sa forme d'édition fait disparaître la structure du module dont elle est issue, pour faire référence essentielle à la sémantique de ce module.

Les algorithmes, développés dans la littérature, et qui traduisent une table en un organigramme, ont montré qu'une table de décisions est la traduction d'un grand nombre d'organigrammes différents. Tous ces organigrammes ont en commun le texte des conditions et assignations et le conditionnement des actions, c'est-à-dire, l'information contenue dans les quatre ensembles :

- Souche conditions,
- souche actions,
- entrées conditions,
- entrées actions.

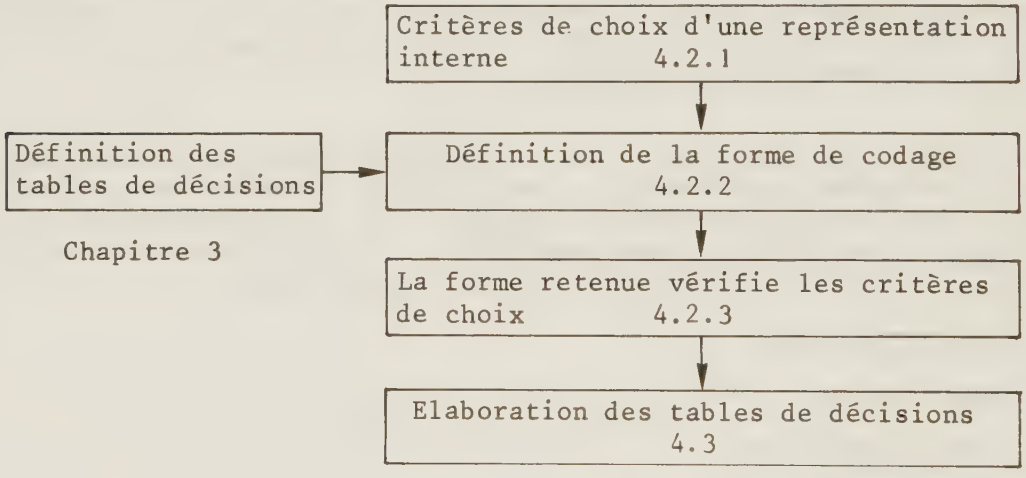
Cette information est caractéristique de la table de décisions. Nous l'appellerons table de décisions abstraite. L'organigramme permet de la retrouver. La table de décisions sous sa forme éditée également. C'est ainsi que nous avons pu définir une table de décisions à l'aide d'un graphe, dont sommets (souches) et arcs (entrées) sont étiquetés. Table éditée et organigramme apparaissent ainsi comme deux graphismes qui représentent le même concept. Selon que l'utilisateur s'intéresse davantage à la structure, ou à la sémantique de la table de décisions abstraite, il utilisera l'une ou l'autre forme. Mais il reste que toute représentation qui permet de caractériser l'information contenue dans la table de décisions abstraite est acceptable.

Celles que nous venons d'évoquer ont été créées pour être lisibles par l'homme, le problème se pose, pour nous, de trouver une représentation accessible à la machine. Ni le graphe même réduit à un dictionnaire des suivants ou à une matrice de connexion, ni la forme tabulaire d'édition ne se prêtent à une mise en mémoire d'ordinateur, intermédiaire indispensable pour notre objet.

Parmi toutes les représentations imaginables de la table de décisions abstraite, nous en choisirons une qui réponde aux besoins que nous formulerons dans ce chapitre. Il sera ensuite possible d'étudier les algorithmes qui permettront la manipulation et l'édition des tables ainsi représentées.

Les éléments indispensables à la traduction d'un programme en tables seront ainsi mis en place. Définition et représentation en mémoire d'une table de décisions à entrées limitées, algorithmes de transformation des tables auront été étudiés. La dernière partie du chapitre utilise ces résultats et décrit les étapes nécessaires à la traduction d'un programme en tables de décisions à entrées limitées. Les algorithmes qui y sont mentionnés seront détaillés dans le chapitre 8 relatif à la chaîne TABOL.

Le schéma suivant résume la démarche logique entreprise dans ce chapitre :



4.2 - FORME CODEE DES TABLES DE DECISIONS

4.2.1 - Critères de choix d'une forme de codage

Parmi toutes les formes de codage qui permettent de résumer l'information contenue dans une table de décisions, il nous faut choisir :

a) Celles qui s'adapteront le mieux à la mise en mémoire d'ordinateur.

b) Et parmi celles-ci, celles qui permettront la plus grande souplesse dans la manipulation des tables de décisions.

c) Enfin, nous chercherons à construire une forme de codage telle que la traduction du langage de programmation soit simple sinon rapide.

a) Adapté à l'ordinateur, le codage ne devra pas utiliser de forme tabulaire, de dimensions toujours imprévisibles, peu maniable et rapidement encombrante. C'est pour cette raison que, ni la forme d'édition ni l'organigramme ne sont un codage acceptable. Il faut remarquer que le programme lui-même, suite hiérarchisée d'instructions contient l'information qui en fait une représentation de tables de décisions. Cette forme a l'avantage d'être assimilable par l'ordinateur. Nous remarquerons qu'elle se présente de façon naturellement séquentielle. Seules la lourdeur et les redondances du langage COBOL nous ont fait écarter cette solution intellectuellement satisfaisante.

b) Le codage devra encore permettre la manipulation des tables de décisions qu'il faudra entreprendre au cours de leur construction. Nous aurons à envisager les opérations suivantes :

- Regrouper deux tables,
- séparer ou singulariser une sous-table,
- éditer une table.

4.2.1.1 - Regrouper deux tables

Nous avons signalé au chapitre précédent que la dernière action entreprise pour chaque règle d'une table de décisions est une rupture de séquence. Les ruptures de séquence pratiquement rencontrées sont ou l'arrêt définitif du programme, ou l'appel à l'exécution d'une table de décisions, différente ou non de la table qui appelle. Nous appellerons regroupement de deux tables l'opération qui consiste à construire une seule table de décisions en contractant souches et entrées de deux tables dont l'une appelle l'autre, par au moins l'une de ses règles. Cette opération à laquelle il sera fait fréquemment allusion dans la suite du texte, joue un rôle prépondérant dans la construction des tables de décisions d'un programme.

Ainsi la table A de l'exemple suivant appelle la table B par l'intermédiaire des règles 2 et 4.

TABLE A

LE CLIENT VEUT UNE PLACE DE PREMIERE CLASSE	Ø	Ø	Ø	N
IL RESTE DES PLACES LIBRES EN PREMIERE	Ø	N	N	
LE CLIENT ACCEPTE DE CHANGER DE CLASSE		Ø	N	
DELIVRER UN BILLET DE PREMIERE	*			
INSCRIRE SUR LA LISTE D'ATTENTE PREMIERE			*	

RUPTURES DE SEQUENCE

FIN	*		*	
VERS TABLE B		*		*

TABLE B

IL RESTE DES PLACES TOURISTES LIBRES	Ø	N	N
LE CLIENT A DEJA ACCEPTE UN CHANGEMENT DE CLASSE		Ø	N
DELIVRER UN BILLET TOURISTE	*		
INSCRIRE SUR LA LISTE D'ATTENTE TOURISTE			*
INSCRIRE SUR LA LISTE D'ATTENTE PREMIERE		*	
RUPTURES DE SEQUENCE			
FIN	*	*	*

Le regroupement des tables A et B a pour effet de remplacer la forme codée de la table A par celle de la table résumée décrite ci-dessous.

TABLE REGROUPEE A + B

LE CLIENT VEUT UNE PLACE DE PREMIERE CLASSE	Ø	Ø	Ø	Ø	Ø	N	N	N
IL RESTE DES PLACES LIBRES EN PREMIERE	Ø	N	N	N	N			
LE CLIENT ACCEPTE DE CHANGER DE CLASSE		Ø	Ø	Ø	N			
IL RESTE DES PLACES LIBRES TOURISTES		Ø	N	N		Ø	N	N
LE CLIENT A DEJA ACCEPTE UN CHANGEMENT DE CLASSE			Ø	N			Ø	N
DELIVRER UN BILLET DE PREMIERE	*							
INSCRIRE SUR LA LISTE D'ATTENTE PREMIERE			*		*		*	
INSCRIRE SUR LA LISTE D'ATTENTE TOURISTE				*				*
DELIVRER UN BILLET TOURISTE		*				*		
RUPTURES DE SEQUENCE								
FIN	*	*	*	*	*	*	*	*

4.2.1.2 - Sous-tables

La notion de sous-tables a été définie au chapitre précédent. Nous y avons souligné qu'une sous-table de décisions est entièrement déterminée par la donnée de son titre. La forme codée d'une table de décisions devra permettre de singulariser rapidement une sous-table, soit la sous-table entière, soit son titre. Cette propriété de la forme codée sera utilisée lors de la recherche des ruptures sémantiques dans la table. Nous devrons encore réservé la possibilité de construire simplement le codage des deux tables séparées : table principale et sous-table.

4.2.1.3 - Edition des tables

A l'évidence la forme de codage retenue devra permettre l'édition aussi simple que possible des tables de décisions. L'algorithme d'édition est l'algorithme principal de notre étude. Destiné à produire une représentation proche de l'homme, il faut prévoir que les seuls impératifs de l'édition : mise en page, etc. le rendront assez lourd.

Nous imposerons que le même programme édite tables à entrées limitées et à entrées étendues. Et par là, nous imposons que la forme codée que nous recherchons puisse être rapidement généralisée à la représentation des tables à entrées étendues. Il en résultera que les tables de décisions à entrées mixtes ne présenteront pas un cas particulier.

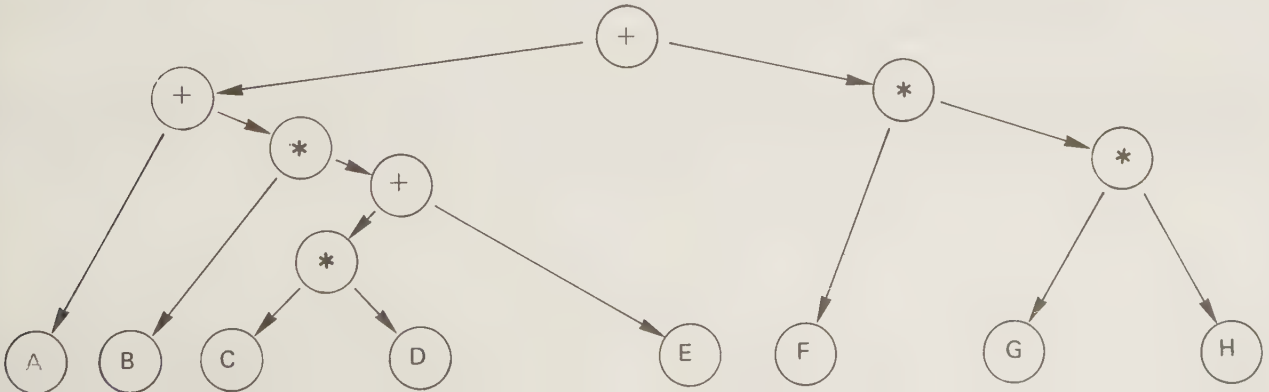
4.2.2 - Définition du codage

4.2.2.1 - Arborescence binaires

Il est classique d'associer à un graphe arborescent binaire une forme préfixée d'expression, arithmétique par exemple. Les sommets du graphe de degré extérieur 2 sont étiquetés par un opérateur binaire, les sommets pendants de l'arborescence sont étiquetés par un opérande. Ainsi à l'expression suivante :

A + B * (C * D + E) + F * G * H

peut être associée l'arborescence :



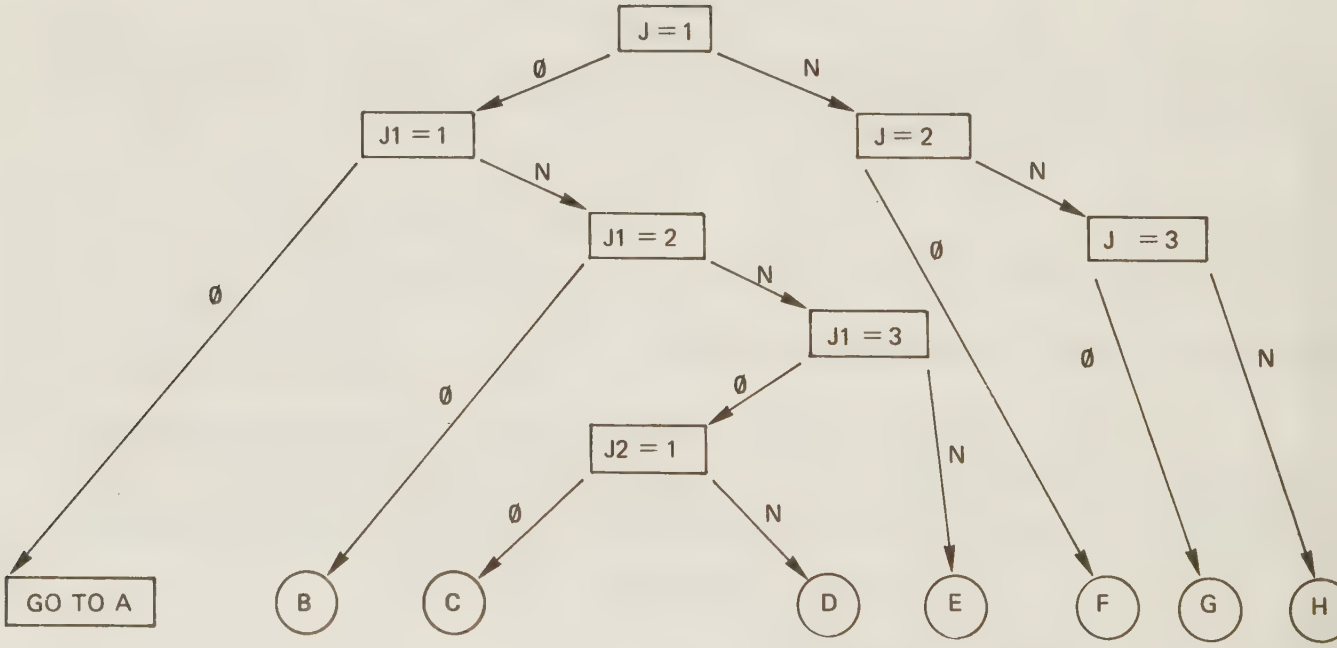
et la forme préfixée ou polonaise :

+ + A * B + * C D E * F * G H

Notre propos n'est pas de détailler la construction de cette forme préfixée, mais d'exploiter la correspondance entre graphe et forme polonaise, mise en évidence par cet exemple.

4.2.2.2 - Première grammaire

Nous avons montré au chapitre précédent qu'un graphe arborescent binaire permet de représenter certaines tables de décisions à entrées limitées. Les sommets étiquetés par un opérateur arithmétique dans l'exemple précédent sont alors étiquetés par une condition, les sommets pendants sont étiquetés par une rupture de séquence. Le graphe devient une table de décisions à condition que l'on en précise les entrées. Par convention, nous étiquetterons l'arc de gauche descendant d'une condition par Ø, l'arc de droite par N. Soit par exemple :



La forme préfixée devient alors :

J = 1, J1 = 1, GO TO A, J1 = 2, GO TO B, J1 = 3, J2 = 1, GO TO C, GO TO D, GO TO E,
J = 2, GO TO F, J = 3, GO TO G, GO TO H

Cette forme peut être considérée comme un codage de la table de décisions représentée par l'organigramme et dont la forme éditée est dessinée ci-dessous. La lecture de la forme codée montre qu'il paraît opportun de séparer textes des souches et codage de la table, en remplaçant dans la forme codée un texte, par un nombre qui lui soit propre, pour rassembler les textes dans un fichier indépendant.

La forme ainsi construite apparaîtra comme une succession de nombres, clés d'accès aux textes de souches. Le rôle de séparateur, joué ici par la virgule peut alors être omis. Nous en conserverons cependant un, qui par mesure d'homogénéité sera un chiffre, pour annoncer la nature : condition, action ou rupture de séquence, du texte associé à la clé qui suit. Par convention le chiffre 7 caractérise une condition, 2 une rupture de séquence.

Avec ces nouvelles conventions, le codage de la table donnée en exemple sera :

7 01 7 02 2 08 7 03 2 09 7 04 7 05 2 10 2 11 2 12 7 06 2 13 7 07 2 14 2 15

La forme éditée de la table correspondante sera :
(Les chiffres en tête de lignes indiquent la référence du texte, utilisée dans la forme codée.)

01	J = 1	Ø	Ø	Ø	Ø	Ø	N	N	N
02	J1 = 1	Ø	N	N	N	N			
03	J1 = 2		Ø	N	N	N			
04	J1 = 3			Ø	Ø	N			
05	J2 = 1			Ø	N				
06	J = 2						Ø	N	N
07	J = 3							Ø	N
08 à 15	GO TO	A	B	C	D	E	F	G	H

Nous avons ainsi construit sur cet exemple une grammaire à structure de phrase qu'il reste à décrire. Le vocabulaire terminal est formé de :

- les caractères 7 et 2,

- les ruptures de séquences et la souche conditions. Nous en noterons les éléments représentatifs respectivement : S et C. Rappelons qu'il s'agit de nombres caractéristiques du texte lui-même.

L'axiome est TDEL.

Les règles sont les suivantes :

Règle 1 TDEL ⟶ 7 C <opérande> <opérande>

Règle 2 <opérande> ⟶ .TDEL

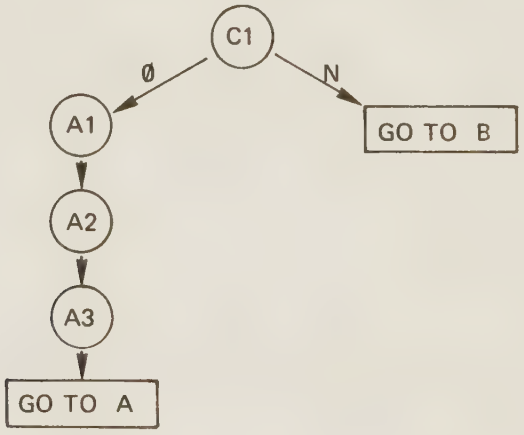
Règle 3 <opérande> ⟶ 2 S

On vérifiera aisément que la forme donnée en exemple est une phrase de la grammaire ainsi définie.

4.2.2.3 - Construction des règles de syntaxe

Dans l'exemple de la table conditionnelle ci-dessus, chaque règle était réduite à sa rupture de séquence ; et la règle de formation numéro 3 a suffi à décrire les opérandes formés par les instructions GO TO.

On peut trouver des structures plus complexes. A une séquence d'actions correspondra un groupe de terme dans le codage. Ainsi la séquence de la règle 1 suivante :



sera codée comme 8 A1 9 A2 9 A3 3 S

La séquence est limitée par un pointeur 8 sur la première action et 3 sur la rupture de séquence.

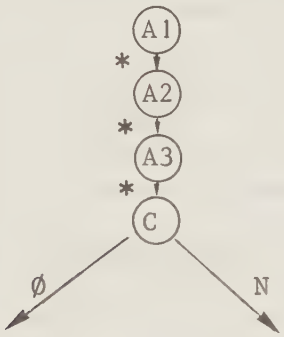
Pour rendre compte de telles éventualités, nous enrichirons le vocabulaire terminal des caractères 8, 9 et 3 et de la souche action de la table dont l'élément représentatif sera noté A. Ils obéiront aux règles :

Règle 3 - 0 <opérande> ⟶ 8 A <séquence>

Règle 3 - 1 <séquence> ⟶ 3 S | 9 A <séquence>

Il apparaît ainsi qu'un opérande de condition est ou réduit à une rupture de séquence pointée par le caractère 2 (Règle 3), ou est une séquence d'actions terminée par une rupture de séquence. Dans ce cas, la première assignation est pointée 8, les suivantes 9 et la rupture de séquence, 3.

Une séquence d'actions peut encore se terminer par une condition. Nous dirons que la séquence est conditionnelle. C'est le cas présenté par la figure 3 au chapitre 3 :



Pour représenter une séquence conditionnelle, nous introduirons le caractère 6 et la règle de formation :

Règle 3 - 2 <séquence> ⟶ 6 C <opérande> <opérande>

La séquence donnée en exemple sera codée par :

8 A1 9 A2 9 A3 6 C ...

Enfin, une table de décisions peut commencer par une séquence d'actions, et non pas seulement par une condition, comme l'impose la règle 1. Nous ajouterons donc la règle de formation suivante :

Règle 1 - 0 TDEL ⟶ <opérande>

4.2.2.4 - Validité de la définition

Les extensions que nous venons d'apporter à la grammaire de codage, permettent de représenter un graphe quelconque à structure de table de décisions. Les différentes structures possibles ont été représentées au chapitre 3 (figures 1, 2, 3 et 4).

Réciproquement, montrons qu'à une phrase de la grammaire, il est possible d'associer un graphe à structure de tables de décisions.

Soient C1, C2, A1, A2, A3 et S1, S2, S3 des clés représentant respectivement, des textes de conditions, actions et ruptures de séquence. La phrase

7 C1 8 A1 3 S1 8 A2 6 C2 2 S2 8 A3 3 S3

est une phrase de la grammaire. Construisons le graphe qu'elle représente.

La première clé déterminera le titre de la table, ici C1.

Les caractères 6 et 7 indiquent un sommet étiqueté par une condition. Un tel sommet aura donc deux suivants (C1 et C2).

Les caractères 8 et 9 permettent de reconnaître une action.

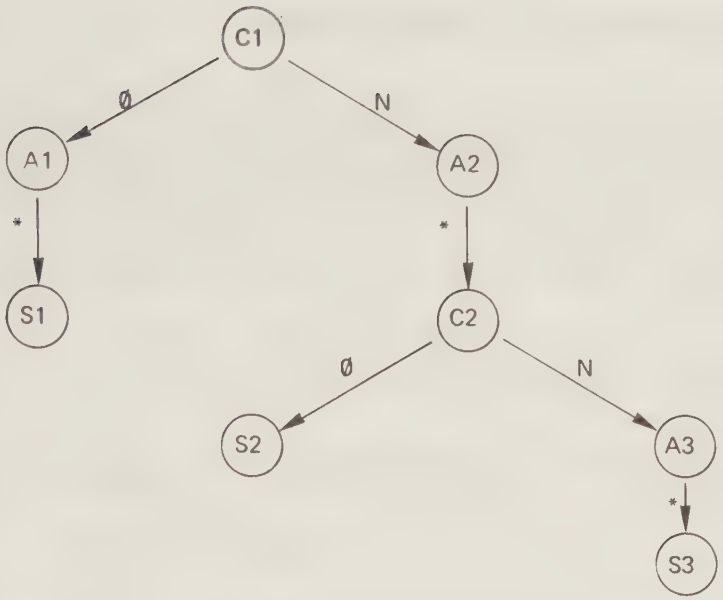
Les caractères 2 et 3 pointent une rupture de séquence.

Par ailleurs :

Les caractères 7, 8 et 2 attachés à un sommet, imposent que son antécédent, s'il existe, soit une condition. Chacun des deux opérandes de la condition est donc repéré par un tel pointeur. Au premier rencontré dans le codage, est associée l'étiquette Ø, au second l'étiquette N. Ces trois caractères apparaissent comme les seuls qui puissent pointer le titre d'une table.

Les caractères 6, 3 et 9 indiquent que l'antécédent du sommet considéré est étiqueté par une assignation. L'arc de jonction porte l'étiquette ✱.

L'application de ces règles de transformations, à l'exemple donné plus haut, permet de tracer l'organigramme équivalent à la phrase de code.



A cette forme correspondra ensuite la forme éditée :

C1	Ø	N	N
C2		Ø	N
A1	*		
A2		*	*
A3			*
S1	*		
S2		*	
S3			*

La transformation que nous avons décrite, permet de conclure à l'équivalence entre l'ensemble des graphes à structure de tables de décisions, et les phrases de la grammaire que nous venons de construire.

Une phrase de cette grammaire et le fichier des souches, que lui associent les clés qu'elle contient, résument entièrement l'information contenue dans le graphe ou la forme éditée de la table. Il s'agit donc là d'une représentation possible de la table de décisions abstraite.

4.2.3 - Propriétés de la forme de codage

Il reste à montrer que cette représentation remplit les objectifs que nous avons imposés, en évaluant ses possibilités dans chacun des domaines annoncés. Le caractère principal de cette forme codée est de se présenter comme une séquence linéaire de termes, qui autorise l'utilisation de simples fichiers séquentiels à l'exclusion de tout tableau, De cette linéarité provient la simplicité des algorithmes de regroupement et séparation que nous allons étudier.

4.2.3.1 - Regroupement de tables

L'algorithme de regroupement est une application particulière de la récursivité qu'indique la règle de formation 2. Rappelons cette règle :

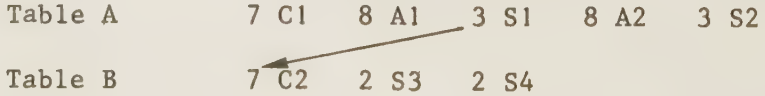
<opérande> ⟶ TDEL

Une table de décisions entière apparaît donc comme un opérande de condition syntaxiquement correct.

Soit le codage d'une table A, tel qu'un opérande se réduise à une rupture de séquence, dont l'effet soit l'appel à l'exécution d'une table B :

La substitution pure et simple de la phrase résumant B et de la rupture de séquence qui l'appelle dans A, aboutit à la formation d'une phrase syntaxiquement correcte. Pour montrer que cette phrase est le codage de la table regroupée AB, il faut se référer aux graphes associés à chaque table. La substitution envisagée a pour effet, de remplacer l'arc incident intérieurement à la rupture de séquence d'appel, par un arc de même extrémité initiale et dont l'extrémité finale est le titre de la table B. L'étiquette qu'il porte reste identique, puisque déterminée par le sommet origine non changé. Le graphe obtenu est bien celui désiré.

Toute rupture de séquence ne porte pas le pointeur 2, c'est-à-dire ne forme pas à elle seule un opérande complet. Si la rupture de séquence d'appel porte le pointeur 3, le codage obtenu par simple substitution n'est plus syntaxiquement correct. Un exemple le montre :



en supposant que le texte associé à S1 appelle la table B. Le codage :

7 C1 8 A1 7 C2 2 S3 2 S4 8 A2 3 S2

n'est pas une phrase de la grammaire. On pourra s'en assurer en cherchant à construire l'organigramme associé. La substitution a créé un opérande supplémentaire. Au contraire la phrase :

7 C1 8 A1 6 C2 2 S3 2 S4 8 A2 3 S2

est syntaxiquement correcte. Le graphe associé est déduit des graphes des tables A et B par la génération d'un arc A1 ⟶ C2. Il s'agit du graphe voulu et le codage correspondant est celui désiré :

Nous avons signalé, en *4.2.2.4* que les seuls pointeurs acceptables pour le titre d'une table sont 7, 8 ou 2. Le regroupement des tables liées par une rupture de séquence pointée 3, dernier terme d'une séquence, doit s'accompagner d'une modification du pointeur du titre de la table appelée :

7	devient	6
8	"	9
2	"	3

En pratique, le regroupement de deux tables seules ne sera jamais envisagé. L'algorithme de regroupement utilisé permet le regroupement de plusieurs tables liées entre elles. La récursivité désirée est prise en compte au moyen d'une pile. Lecture des tables initiales et écriture du codage résultant sont réalisées terme après terme. A la rencontre d'une rupture de séquence, la table

appelée est déterminée. Si la décision de regrouper les deux tables est prise, l'indice qui sert à la description de la table appelante est empilé, le pointeur du titre de la table appelée est construit, enfin la description de la table appelée est engagée en séquence. A la rencontre d'une fin de phrase, la description de la dernière table d'appel est reprise en séquence à partir de la dernière rupture de séquence qui a donné lieu à regroupement, sa position se trouve indiquée par le sommet de la pile. Enfin si celle-ci est vide la table résultante est entièrement construite, le regroupement est achevé.

4.2.3.2 - Sous-tables

La même règle syntaxique 2 :

<opérande> ⟶ TDEL

conduit à considérer tout opérande de condition comme une sous-table de décisions. Les règles de formation 3, 3 - 0, 3 - 1, 3 - 2, imposent que le dernier terme d'un opérande fasse référence à une rupture de séquence. Il apparaît ainsi que les sous-tables, dont le titre est le premier terme d'un opérande, c'est-à-dire est pointé 7, 8 ou 2, sont codées par une succession continue de termes dans le codage de la table principale. Un tel titre étant donné, la recherche du codage de la sous-table correspondante se résume donc à construire une production sémantique de la règle 2. L'algorithme qui permet cette construction sera étudiée dans le paragraphe relatif à la construction de la souche condition : *4.3.3.1.2.*

Nous avons signalé que tout sommet du graphe d'une table de décisions définit une sous-table. La recherche du codage de la sous-table est l'opération inverse de la formation du codage de deux tables regroupées. Si le terme correspondant à un sommet du graphe se trouve être pointé par l'un des caractères 6, 3 ou 9, c'est-à-dire s'il appartient à ce qui a été appelé une séquence, le codage de la sous-table qu'il détermine est encore une succession continue de termes dans le codage de la table principale. Le pointeur du titre sera modifié suivant le tableau :

6	devient	7
3	"	2
9	"	8

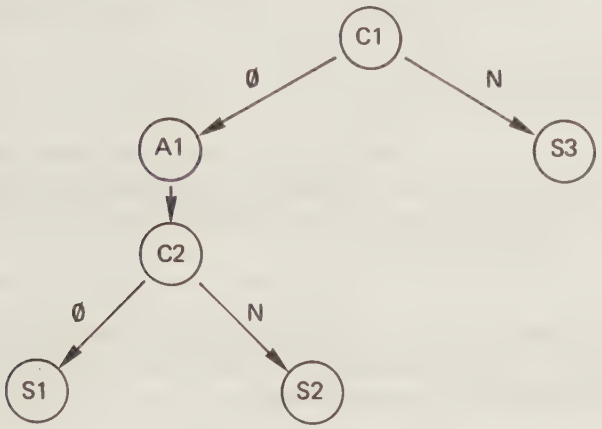
4.2.3.3 - Langage de programmation

Nous n'étudierons pas ici l'algorithme qui construit la forme d'édition d'une table codée. La dernière propriété de cette forme de codage que nous étudierons, concerne la traduction du langage COBOL, qu'il faudra entreprendre pour obtenir une phrase syntaxiquement correcte.

Soit la phrase COBOL du type IF ... THEN ... ELSE ... :

IF C1 THEN A1 IF C2 THEN S1 ELSE S2 ELSE S3

Le graphe correspondant peut être :



et la phrase qui code ce graphe à structure de table de décisions :

7 C1 8 A1 6 C2 2 S1 2 S2 2 S3

Il apparaît que les références aux conditions, actions et ruptures de séquence restent dans le même ordre. Seules les parenthèse IF, THEN, ELSE sont modifiées et remplacées par les caractères de pointage. L'analyse syntaxique du langage COBOL se trouve donc être singulièrement simplifiée. Seules les conditions composées avec les conjonctions OR et AND donneront lieu à une analyse détaillée.

Une conséquence intéressante de cette propriété réside dans le fait que la forme de codage d'une table de décisions peut être interprétée et donc contrôlée de façon relativement simple par référence à la phrase COBOL qu'elle masque.

La sujétion au langage COBOL, des algorithmes et de la méthode générale que nous proposons pour documenter un programme est pratiquement limitée au parallélisme qui vient d'être mis en évidence entre phrase conditionnelle COBOL et forme de codage d'une table. Il en résulte que seule la phase d'analyse syntaxique doit être modifiée pour permettre la documentation d'un programme écrit dans tout autre langage qui conserve la structure de phrase conditionnelle :

IF ... THEN ... ELSE ...

et les imbrications autorisées.

4.2.4 - Forme normale de Bacchus

Nous conclurons cette étude en retenant la forme de codage proposée par les phrases de la grammaire construite, puisqu'elle apparaît compatible avec nos objectifs.

4.2.4.1 - Ruptures sémantiques

Il reste cependant à compléter la définition par les deux règles suivantes qui permettront de rappeler une rupture sémantique. Ce phénomène, défini en *3.4.2.1*, peut être traduit par la singularisation de la sous-table dont le titre est le sommet étiqueté par la condition, objet de la rupture.

Ainsi le codage rendra compte de la rupture par la seule particularisation du pointeur affecté à la condition. En pratique les pointeurs 6 et 7 caractéristiques d'une clé de condition deviendront respectivement 4 et 5.

Ceci peut être introduit par les règles de syntaxe :

Règle 3 - 3 <opérande> ⟶ 5 C <opérande> <opérande>

Règle 3 - 4 <séquence> ⟶ 4 C <opérande> <opérande>

4.2.4.2 - Nom et titre de table

Enfin, nous imposerons à une table de décisions d'avoir un nom qui sera édité au début de la table. Ce nom pourrait être l'étiquette portée par le titre de la table, mais outre que ce nom ne serait pas unique, il obligerait de modifier le texte des ruptures de séquence, qui en COBOL mentionnent un nom de paragraphe. Il a paru raisonnable d'imposer que le titre d'une table soit toujours la première instruction d'un paragraphe du programme COBOL. Le nom de la table sera naturellement le nom du paragraphe correspondant. Les noms de paragraphes sont mémorisés dans un fichier particulier et nous retiendrons dans le codage l'adresse du nom dans le fichier. Le pointeur caractéristique de cette adresse sera 0. Le vocabulaire terminal est enrichi de l'ensemble des adresses P de noms de table et de la parenthèse 0000.

Les règles de formation sont :

Règle 1 TDEL ⟶ 0 0000 0 P 0 0000 <TD> 0 0000

Règle 1 - 0 <TD> ⟶ 7 C <opérande> <opérande>

Règle 1 - 1 <TD> ⟶ <opérande>

4.2.4.3 - Conclusion

En résumé la grammaire ainsi construite admet pour vocabulaire terminal :

L'ensemble $\mathcal{C}$ des clés d'accès C à la souche condition.
L'ensemble $\mathcal{A}$ des clés d'accès A à la souche condition.
L'ensemble $\mathcal{S}$ des clés d'accès S aux ruptures de séquence.
L'ensemble $\mathcal{P}$ des clés d'accès P aux noms des tables.
La parenthèse 0000.
Les caractères 0, 2, 3, 4, 5, 6, 7, 8, 9.

L'axiome est TDEL et les règles de formation :

R1 TDEL ::= 0 0000 0 P 0 0000 <TD> 0 0000

R2 <TD> ::= 7 C <opérande> <opérande> | <opérande>

R3 <opérande> ::= <TD>

R4 <opérande> ::= 2 S | 8 A <séquence> | 5 C <opérande> <opérande>

R5 <séquence> ::= 3 S | 9 A <séquence> | 6 C <opérande> <opérande> |
4 C <opérande> <opérande>

4.3 - ELABORATION DES TABLES DE DECISIONS A ENTREES LIMITEES D'UN PROGRAMME COBOL

Après avoir décrit la nature et la forme des tables de décisions que nous utilisons, nous décrirons ici le traitement que subit un programme COBOL. Ce traitement est pratiquement réalisé par les programmes de la chaîne TABOL.

Les paragraphes qui suivent n'offrent pas une étude détaillée des différents algorithmes utilisés en pratique mais une présentation générale des étapes nécessaires à l'élaboration automatique des tables de décisions.

La présentation des algorithmes les plus originaux, utilisés dans la réalisation pratique à laquelle cette étude a donné lieu, fait l'objet du chapitre 8. La distinction de ces degrés de détail paraîtra arbitraire dans quelques cas. Nous prions le lecteur de bien vouloir excuser les éventuelles répétitions qu'il pourrait trouver.

Il résulte de la définition d'une table de décisions que seul le titre peut être de degré intérieur supérieur à 1. A chaque sommet du graphe d'un programme ayant plus d'un antécédent, devrait correspondre une table. Nous avons montré qu'une telle démarche n'est pas convaincante. Les tables construites auraient de trop petites dimensions. La construction des tables de décisions est donc plus que la recherche dans le graphe du programme des sous-graphes à structure de tables de décisions.

Nous distinguerons les trois phases suivantes dans l'élaboration des tables de décisions :

4.3.1 - Recherche des sous-graphes à structure de tables de décisions, réalisée par le programme PD1B de la chaîne.

4.3.2 - Construction des tables de décisions par transformation du graphe du programme PD1C et PD2A.

4.3.3 - Edition des tables de décisions à entrées limitées. Programme PD3A et PD3B.

4.3.1 - Recherche des sous-graphes à structures de tables - Tables atomiques

La première étape de l'étude des procédures du programme donné a pour effet de construire un jeu de tables de décisions. Chaque phrase conditionnelle ou chaque séquence d'instructions inconditionnnelles, est codée sous la forme d'une table indépendante que nous appellerons table atomique.

La justification de cette démarche est déduite des propriétés déjà étudiées d'une table de décisions, à savoir :

4.3.1.1 - Une suite d'instructions en séquence est une table de décisions inconditionnelle.

4.3.1.2 - Conditions composées

Une phrase conditionnelle COBOL est une table de décisions. La présence de conditions composées à l'aide de conditions simples liées par les conjonctions OR et AND et de la négation NOT, conduit à une étude analytique complète de la phrase. Il s'agit de construire le graphe arborescent équivalent à la condition

composée. Ainsi :

IF C1 OR C2 THEN oui ELSE non

est traduit par le code équivalent à la phrase COBOL :

IF C1 THEN oui ELSE IF C2 THEN oui ELSE non

4.3.1.3 - Instructions conditionnelles

Les instructions conditionnelles qui résultent de l'emploi des mots et expressions réservées : AT END, INVALID KEY, DEPENDING ON, UNTIL, ON SIZE ERROR font l'objet d'une traduction particulière. Elles conduisent à des phrases conditionnelles standard, générées à partir de textes de conditions, conventionnels. Le détail en est décrit au chapitre 8, au cours de l'étude du programme PD1B. Le résultat est dans tous les cas une table de décisions atomique.

4.3.2 - Graphe du programme

Au cours de l'étape suivante les tables atomiques vont être regroupées entre elles. Nous obtiendrons ainsi des tables de décisions, de dimensions suffisamment importantes, pour offrir une documentation synthétique. Le regroupement est réalisé en deux étapes successives.

4.3.2.1 - Tables de paragraphe

Nous avons imposé que le titre d'une table de décisions, telle qu'elle sera éditée, soit la première instruction d'un paragraphe COBOL. La table reçoit le nom du paragraphe. Les tables atomiques d'un même paragraphe du programme sont regroupées en une seule table, ou table de paragraphe. L'algorithme utilisé a été décrit.

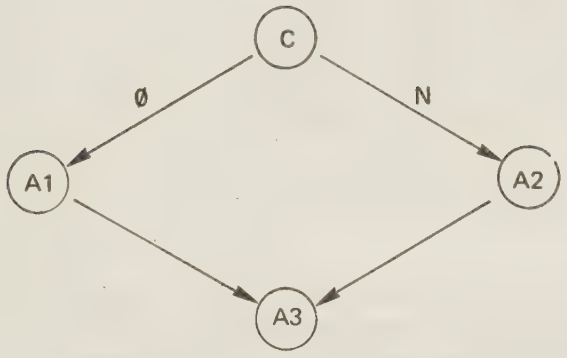
Ce premier passage de regroupement, est équivalent à une transformation du graphe du programme. Soit l'exemple déjà cité :

```
P1
      IF  C THEN  A1  ELSE  A2 .
      A3
```

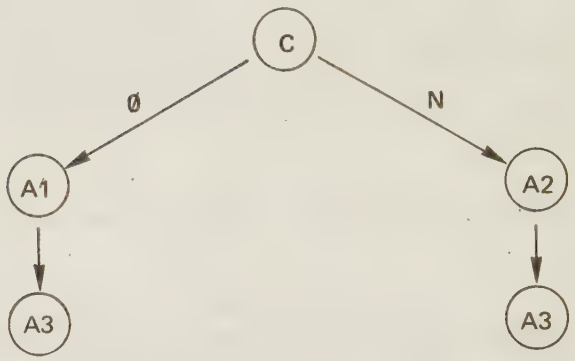
Un tel paragraphe donne lieu à la naissance de deux tables atomiques. Leur regroupement permet de construire une table équivalente à :

IF C THEN A1 A3 ELSE A2 A3 .

Le graphe était :



Il est devenu :



Chaque table atomique qui présente plusieurs antécédents est ainsi répétée sur chacune des branches qui l'appellent.

Au cours de cette première phase de regroupement, le graphe du programme est construit. Chaque sommet en est étiqueté par un nom de paragraphe. Un arc (P1, P2) est créé si le paragraphe P1 appelle le paragraphe P2 au moyen d'une instruction GO TO ou d'un appel en séquence. Les arcs (P1, P2) et (P3, P1) sont générés s'il est trouvé dans P1 une instruction :

PERFORM P2 THRU P3

4.3.2.2 - Entrées de circuits - Noeuds de losanges

L'édition d'une table de décisions pour chacun des paragraphes du programme peut conduire à une documentation trop analytique. La deuxième phase de regroupement est activée sur option donnée par l'utilisateur. Elle permet de regrouper certaines tables de paragraphes à toutes celles qui l'appellent. Avant d'étudier ce deuxième regroupement nous chercherons un critère qui permette d'élaborer la décision de regrouper ou non, deux tables de paragraphes liées entre elles. Ce critère est déduit de l'étude des circuits du graphe G du programme.

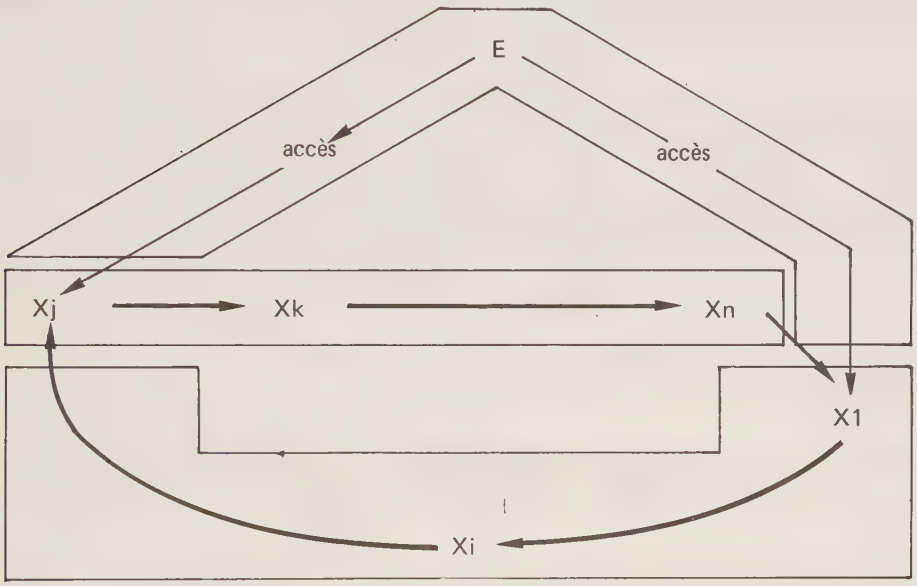
Le graphe G est connexe. Soit E l'étiquette portée par l'unique sommet d'entrée du graphe et soit un circuit dans G :

(X1, X2 ... ,Xn, X1)

où les Xi sont étiquetés par un paragraphe. Le graphe étant connexe, il existe au moins un chemin d'extrémité initiale E et dont l'extrémité finale est un sommet du circuit. Soit M l'ensemble de ces chemins. Nous appellerons accès au circuit, les chemins de M dont seule l'extrémité finale appartient au circuit. Nous appellerons entrées de circuits les sommets du circuit, extrémités finales des accès. Une en-

trée de circuit a au moins deux antécédents. L'un appartient au circuit, l'autre à l'accès.

Nous imposerons qu'un tel sommet soit le titre d'une table éditée. Pour cela la décision de regrouper une table T2 sur les règles d'une table T1 qui l'appellent sera prise, si et seulement si le titre de T2 n'est pas une entrée de circuit. Le schéma montre que les tables représentant les accès se rompent sur le circuit. Le circuit lui-même est traduit par des tables représentant une partie de graphe comprise entre deux accès.



Les entrées de circuits que nous venons de définir ne sont pas les seuls sommets donnant lieu à rupture des tables. Les paragraphes référencés par PERFORM seront considérés comme des modules indépendants. L'entrée de chaque module appelé par PERFORM deviendra le titre d'une table de décisions éditée. Il a paru, enfin, utile de laisser à l'utilisateur, la possibilité de désigner des paragraphes, comme entrées de modules logiques de traitement. Il leur correspondra également un titre de table de décisions.

En résumé, le critère de regroupement de deux tables peut s'énoncer :

Le regroupement de la table de paragraphe T2 sur les règles de la table T1 qui l'appellent est entrepris si et seulement si :

- la table résultante n'est pas de dimensions trop grandes,

- T2 n'est pas une entrée de circuit, une tête de module appelée par perform ou désignée par l'utilisateur.

Le regroupement des tables atomiques et la détermination des entrées de circuits du graphe G sont réalisés par le programme PD1C de la chaîne TABOL.

4.3.2.3 - Regroupement des tables de paragraphes

Cette deuxième étape du regroupement est réalisée par le programme PD2A. Les entrées de circuits du programme ne sont pas les seuls sommets dont le degré intérieur est supérieur à 1. Nous appellerons noeud de losange un sommet du graphe G dont le degré intérieur est supérieur à 1, et qui n'est pas une entrée de circuit. Le regroupement de deux tables est entrepris si la table appelée est de degré intérieur 1 ou si c'est un noeud de losange.

Le regroupement sur toutes les tables qui l'appellent, d'une table associée à un noeud de losange, se traduit sur le graphe G par une transformation qui conserve la logique du programme. Soit n le degré intérieur d'un noeud de losange L. Les regroupements opérés en L sont équivalents à la répétition n-1 fois du sous-graphe à structure de table dont le titre est le noeud de losange L.

4.3.3 - Edition des tables de décisions

Les tables de décisions qui représentent le programme donné sont maintenant entièrement construites. Elles sont mises en mémoire à l'aide de trois fichiers,

Le premier contient les noms des tables de décisions qui seront éditées. Ce sont des noms de paragraphe du programme COBOL.

Le second fichier contient les textes de souches sous forme d'instructions COBOL.

Enfin le dernier contient le codage des tables de décisions, succession de phrases de la grammaire décrite dans ce chapitre. Le codage est une suite de nombres qui référencent les articles des deux fichiers précédents.

Nous étudierons, maintenant, les algorithmes qui permettent l'édition d'une table de décisions sous la forme habituelle. L'édition est préparée par le programme PD3A de la chaîne. Elle est réalisée par le programme PD3B.

Nous décrirons tour à tour la construction des éléments constitutifs de la table éditée :

- souche conditions,
- souche actions,
- entrées conditions,
- entrées actions.

4.3.3.1 - Souche conditions

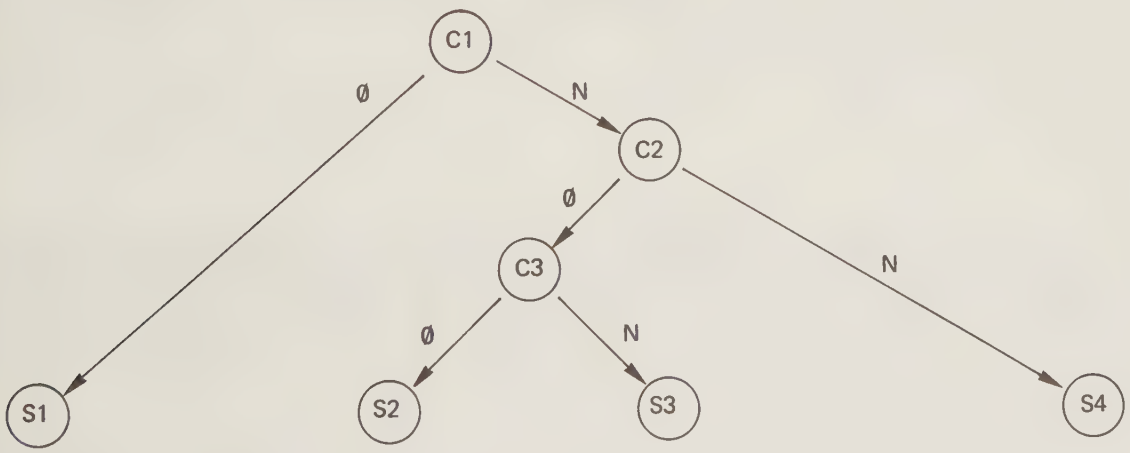
La construction de la souche condition de la table éditée résulte de la comparaison entre eux des textes de toutes les conditions qui interviennent dans une même sous-table d'édition. L'ordre d'édition des textes de conditions distinctes peut alors être arbitrairement imposé ; en pratique, on choisira l'ordre des premières occurences dans la phrase de codage.

4.3.3.1.1 - Conditions identiques

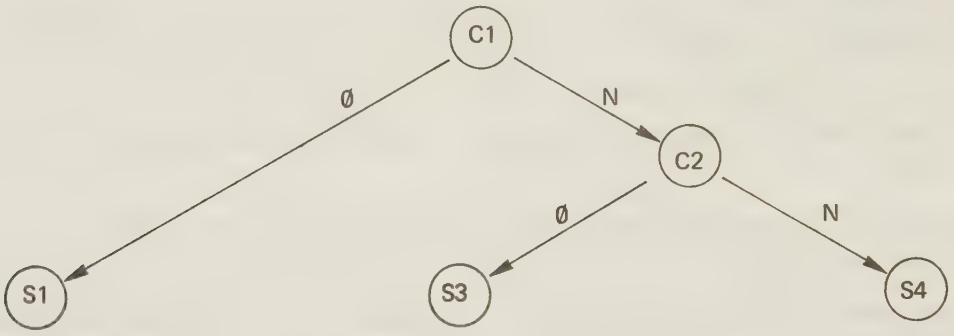
Dans le cas où des textes de conditions sont trouvés identiques une vérification doit être entreprise. Nous introduirons la nécessité de cette vérification par l'exemple de la phrase suivante :

7 C1 2 S1 7 C2 7 C3 2 S2 2 S3 2 S4

représentée par le graphe :



Si les textes attachés à C1 et C3 sont identiques, la règle 2 est inaccessible. Le graphe, sémantiquement équivalent, et correct devient



et la phrase de codage : 7 C1 2 S1 7 C2 2 S3 2 S4

De façon générale, pour chaque condition C, il faut d'abord déterminer la sous-table qu'elle induit, c'est-à-dire dont C est le titre. La recherche de cette sous-table est détaillée au paragraphe *4.3.3.1.2* suivant. Dans la sous-table ainsi déterminée, il ne doit pas y avoir d'occurence de condition identique à C. Dans le cas contraire, une branche est inacessible. Une transformation du graphe équivalent doit être entreprise.

4.3.3.1.2 - Dernier suivant - Suivant non

L'algorithme de vérification décrit au paragraphe précédent, et le détail de plusieurs autre algorithmes que nous présentons par la suite demande que l'on puisse répondre à une question du type :

- Quelle est la phrase qui représente la sous-table de décisions dont le titre est donné ?

Nous avons montré que cette phrase est une suite continue de terme dans le codage de la table entière. La réponse à la question posée consiste à trouver le dernier terme du codage cherché. Nous l'appellerons dernier suivant. A une action est attaché le dernier terme de la séquence à laquelle elle appartient. A une condition est attaché le dernier terme de la branche Non, à laquelle le contrôle est passé si la condition n'est pas vérifiée. Nous appellerons par la suite cette branche suivant Non.

La position du dernier suivant de chaque terme est recherchée systématiquement dans la phrase dès son chargement en mémoire centrale. Soit par exemple la phrase de codage :

7 C1 7 C2 8 A1 6 C3 8 A2 9 A3 3 S1 8 A4 3 S2 8 A5 6 C4 8 A6
9 A7 3 S1 8 A8 3 S2 2 S4

La lecture de droite à gauche de la phrase codée est entreprise. Une pile, chargée au cours de la lecture, mémorise les positions des ruptures de séquence, rencontrées dans la phrase.

A la lecture d'une action, caractérisée par un pointeur 8 ou 9, le sommet de la pile indique la dernière rupture de séquence, c'est-à-dire la fin de la séquence recherchée.

Une condition pointée par les caractères 4, 5, 6 ou 7 conduit à dépiler un premier terme ; celui-ci indique la position de la fin de la dernière séquence attachée à la branche Oui de la condition. Le sommet de la pile indique ensuite la position du dernier suivant cherchée : L'exemple cité conduit à la construction de la troisième ligne. La première indique les positions.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
7 C1	7 C2	8 A1	6 C3	8 A2	9 A3	3 S1	8 A4	3 S2	8 A5	6 C4	8 A6	9 A7	3 S1
17	16	9	9	7	7		9		16	16	14	14	

15	16	17
8 A8	3 S2	2 S4
16		

L'algorithme que nous venons de décrire forme un reconnaisseur syntaxique de la grammaire proposée. L'utilisation d'une pile permet de réduire la récursivité signalée dans la définition d'une phrase :

<TD> ::= 7 C <opérande> <opérande>

<opérande> ::= <TD>

4.3.3.2 - Souche actions

Au contraire des conditions, les textes d'actions doivent être édités dans l'ordre induit par la relation binaire décrite en 3.3.2, lors de la définition de la souche action. Les graphes G_a et G_a^* qui y ont été définis sont pratiquement construits et mis en mémoire à l'aide de leur matrice de connexion.

Soit P la matrice de connexion de G_a. Nous en noterons l'élément générique p_{kl}. A chaque ligne k et chaque colonne l de P correspond une assignation de l'ensemble S_a des actions qui interviennent dans une sous-table d'édition.

. p_{kl} vaut 1 si l'assignation A_k est un antécédent de l'assignation A_l, c'est-à-dire s'il existe dans le graphe de la table un arc (A_k, A_l), p_{kl} vaut zéro dans le cas contraire.

La matrice de connexion Q dont les éléments indiquent les chemins dans le graphe G_a, est alors calculée à partir de P. q_{kl} vaudra 1 s'il existe un chemin (A_k, A_l) dans G_a. Le calcul qui permet de construire Q est classique, il consiste à évaluer les puissances logiques successives de la matrice P.

La règle de simplification qui permet de supprimer les actions dont le texte est identique, peut se traduire sur Q :

Règle

Pour deux sommets identiques A1 et A2 de G_a tels que :

$$q_{12} = q_{21} = 0$$

la ligne et la colonne correspondantes à A2 sont supprimées dans Q de la façon suivante :

a) Si $q_{1i} = 0$ et $q_{2i} = 0$ alors q_{1i} reste nul ;

si $q_{1I} = 1$ alors q_{1i} reste identique ;

si $q_{1i} = 0$ et $q_{2i} = 1$ alors q_{1i} vaut 1.

b) La transformation *a)* est appliquée aux termes q_{i1} de la colonne correspondante à A_1.

c) La ligne et la colonne correspondante à A_2 sont effacées. L'action A_2 est supprimée.

Cette règle est appliquée à tous les couples d'assignations identiques. Soit Q^* la matrice de connexion finalement obtenue. Elle caractérise la relation d'ordre partiel R qui a été définie au chapitre 3. A chaque ligne de Q^* est associée une assignation de la souche action.

En pratique chaque sous-table distinguée à l'édition donne lieu à une étude séparée. Pour chaque matrice Q^* successivement construite, l'ordre d'édition des textes est établi, par application de la règle :

- Une action est choisie si la ligne correspondante dans Q^* ne contient pas de 1. Ligne et colonne correspondante de Q^* sont supprimées.

La relation d'ordre utilisée est partielle. Il en résulte que plusieurs lignes peuvent satisfaire au même moment, la condition de choix. En cas de conflit les actions sont éditées dans l'ordre de première occurence dans la phrase de code.

4.3.3.3 - Entrées conditions

La table est éditée ligne par ligne, dans l'ordre croissant des sous-tables qui doivent être distinguées sur la forme éditée. Pour une référence C à un texte d'instruction conditionnelle, le traitement qui fournit les entrées de la ligne condition correspondante, peut se résumer en trois phases essentielles :

- recherche de tous les termes de codage qui référencent C ;

- pour chacun d'eux, séparation des deux branches Oui et Non de la condition. Le titre de la branche Oui suit immédiatement le terme C dans le codage. Le dernier suivant de la branche Oui a été évalué. Le titre de la branche Non suit immédiatement ce dernier suivant. C'est une conséquence de la règle syntaxique : <TD> ::= 7 C <opérande> <opérande> ;

- dans chacune des deux sous-tables Oui et Non, on recherche les ruptures de séquences. A chacune d'elles correspond une règle. L'entrée de cette règle sur la ligne C est ainsi déterminée.

Ainsi l'édition de la ligne correspondant à la condition C2 dans la phrase :

7 C1 8 A1 6 C2 8 A2 9 A3 3 S1 2 S2 7 C3 7 C2 8 A2 3 S2 8 A4 3 S3 2 S1
peut être détaillée comme suit :

a) détermination des derniers suivants. La première ligne indique les positions :

1	2	3	4	5	6	7	8	9	10	11	12	13	14
7 C1	8 A1	6 C2	8 A2	9 A3	3 S1	2 S2	7 C3	7 C2	8 A2	3 S2	8 A4	3 S3	2 S1
14	7	7	6	6	R1	R2	14	13	11	R3	13	R4	R5

b) la condition C2 est repérée en 3 dernier suivant : 7

- o la branche Oui est écrite à partir de 4 (3+1) et jusqu'à son dernier suivant soit 6 ;
- o la branche Non est formée du seul terme 7 ; de 6+1 à 7 ;

c) la règle de la branche Oui (ici unique) contiendra un Ø sur la ligne de C2. Il s'agit de la règle 1 repérée en 6. De même la règle de la branche Non règle 2 repérée en 7 contiendra un N.

Cette première occurence de la condition C2 conduit à la ligne :

	R1	R2	R3	R4	R5
C2	Ø	N			

De même la deuxième occurence repérée en 9 conduira à la construction de la ligne qui sera finalement éditée :

C2	Ø	N	Ø	N

La partie condition complète sera éditée ligne par ligne pour obtenir :

C1	Ø	Ø	N	N	N
C2	Ø	N	Ø	N	
C3			Ø	Ø	N

4.3.3.4 - Entrées actions

En pratique, c'est le même algorithme qui est employé dans le programme PD3B, pour éditer entrées conditions et entrées actions.

La ligne de l'action A1 de l'exemple précédent est construite comme suit : la référence A1 est repérée en 2. Son dernier suivant est en 7. Toutes les règles dont la rupture de séquence est trouvée dans les positions entre 2 et 7 auront une entrée * sur la ligne de A1, ici les règles 1 et 2.

La partie action éditée sera ici :

A1	*	*			
A2	*		*		
A3	*				
A4				*	

RUPTURES DE SEQUENCE

S1	*				*
S2		*	*		
S3				*	

La forme tabulaire éditée n'est jamais entièrement en mémoire. Le mode de représentation interne choisi a permis de limiter la place nécessaire au traitement d'une table, sans compliquer exagérément les algorithmes de manipulation et d'édition.

4.4 - CONCLUSION

Le traitement d'un programme COBOL donné peut se résumer en trois phases d'élaboration successives :

- La première phase analyse les phrases du programme COBOL. Au cours de cette phase, on crée les trois fichiers fondamentaux qui représentent l'information contenue dans les tables de décisions abstraites. Le fichier contenant les codages, reçoit d'abord les phrases de code des tables atomiques, puis des tables de paragraphe.

- Au cours de la seconde phase, il reçoit le codage des tables qui seront éditées. Ce nouveau cycle est obtenu en regroupant les tables de paragraphes qui correspondent à des noeuds de losanges sur les tables qui représentent des entrées de circuits.

- La dernière phase édite sur l'imprimante les tables de décisions sous leur forme habituelle, décrite au chapitre 2.

CHAPITRE 5

TABLE DE DÉCISIONS A ENTRÉES ÉTENDUES

Les tables de décisions à entrées limitées offrent une documentation synthétique et précise de la forme dans laquelle a été codée un programme. Les textes de souches sont formés des textes même des instructions COBOL. Ils n'indiquent pas clairement, pour le non programmeur, les concepts algorithmiques codés par telle comparaison ou tels mouvements de valeurs.

Pour augmenter l'efficacité de la documentation éditée, il peut paraître intéressant de transformer les textes de souches et les entrées d'une table en textes du langage courant. L'édition de tables de décisions à entrées étendues sera alors dirigée par l'utilisateur. Au moyen de directives convenables, on pourra créer des tables qui en perdant la précision du langage de programmation gagneront la clarté permise par l'emploi d'un langage moins ésotérique.

Ce deuxième cycle de tables éditées pourra à son tour être repris et transformé. Ainsi de proche en proche les tables à entrées étendues éditées pourront devenir plus synthétiques en s'éloignant du détail de la programmation.

Pour introduire le concept de table de décisions à entrées étendues, nous reprendrons le cheminement logique entrepris à propos des tables de décisions à entrées limitées.

La première étape consiste à définir l'objet représenté par une table à entrées étendues. Nous chercherons ensuite une forme de mise en mémoire, ou forme de codage, acceptable. Elle sera déduite de celle qui a été adoptée au chapitre précédent. Enfin, nous aurons à décrire la construction des tables à entrées étendues en détaillant dans chaque cas le traitement subi par une table qui est l'objet d'une directive de transformation.

5.1 - DEFINITION ET REPRESENTATION DES TABLES DE DECISIONS A ENTREES ETENDUES

La définition des tables de décisions à entrées limitées a été introduite à l'aide d'une représentation de graphes binaires, c'est-à-dire dont les sommets sont de degré extérieur maximum 2. La définition et la manipulation des tables de décisions à entrées étendues, seront liées de la même manière, au codage de graphes non binaires. Ce codage et le graphisme finalement édité seront une nouvelle mise en forme, originale et orientée vers l'utilisateur, de l'information contenue dans les quatre ensembles :

- souche conditions
- souche actions
- entrées conditions
- entrées actions

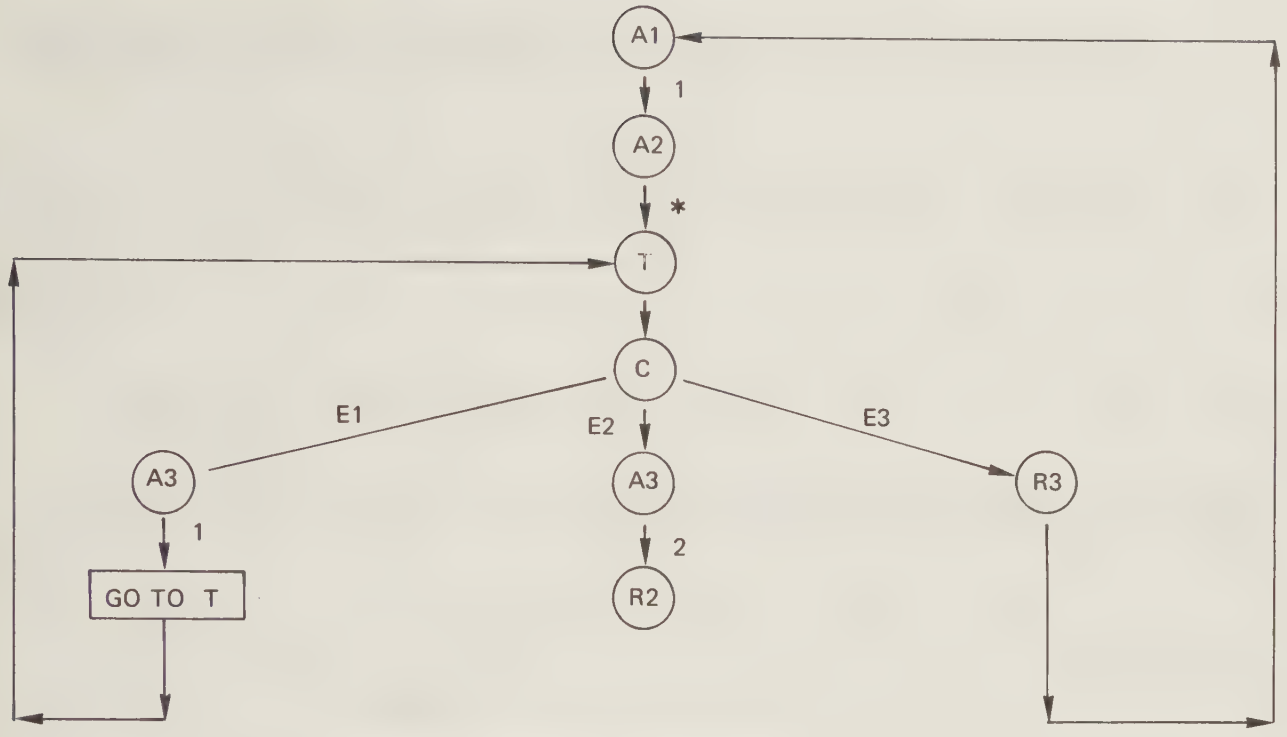
définissant la table de décisions abstraite de référence.

5.1.1 - Sous-tables titrées

La définition d'une table à entrées étendues doit s'attacher à autoriser une grande efficacité dans la documentation. Pour permettre une documentation plus synthétique, nous admettrons que les ruptures de séquences d'une table puissent faire référence à une sous-table de la table. La sous-table est éditée avec un nom et est dite sous-table titrée.

La notion de sous-table a été définie dans le cadre des tables à entrées limitées. Le résultat essentiel est que chaque sommet d'une table peut être le titre d'une sous-table.

L'extension que nous apportons ici intéresse une nouvelle classe de sommets dans le graphe de la table. Un sommet du graphe pourra être étiqueté par une chaîne de caractères, interprétée comme le nom de la sous-table dont il est le titre. Ainsi le graphe suivant pourra définir une seule table de décisions à entrées étendues. En l'absence de l'extension que nous apportons il aurait fallu deux tables distinctes pour représenter une telle structure. La documentation obtenue sera plus groupée.



Le titre d'une table, tel qu'il a été défini pour une table de décisions à entrées limitées perd sa propriété essentielle d'appartenir à tous les circuits du graphe. Une forme éditée de la table dessinée ci-dessus pourrait être :

SOUS-TABLE T

C	E1	E2	E3
A1	1	1	1
A2	*	*	*

SOUS-TABLE T

A3	1	2	
GO TO T	*		
R2		*	
R3			*

L'expérience montre qu'un graphe définissant un programme complet peut être représenté par une table de décisions telle que nous allons la définir.

5.1.2 - Définition d'une table de décisions à entrées étendues

5.1.2.1 - Définition

Nous associerons le nom de table de décisions à entrées étendues à un graphe connexe, fini dont les sommets jouissent des propriétés suivantes :

- Il existe un seul sommet d'entrée. Par analogie nous dirons qu'il s'agit du titre de la table, l'étiquette qu'il porte sera le nom de la table. Il est de degré extérieur 1.

- Un sommet du graphe de degré extérieur supérieur à 1 est appelé "condition". Il est de degré intérieur 1, La chaîne de caractères formée de la concaténation :

  - d'une part de l'étiquette portée par ce sommet,
  - d'autre part de l'une quelconque des étiquettes portées par les arcs incident extérieurement au sommet exprimé, pour l'utilisateur, la valeur Ø d'une prédication.

- Un sommet du graphe de degré extérieur 1 et de degré intérieur supérieur à 1 est appelé titre de sous-table, L'étiquette qu'il porte est le nom de la sous-table qu'il induit.

- Un sommet du graphe de degré extérieur et de degré intérieur égaux à 1 dont le suivant n'est ni le titre de la table, ni un titre de sous-table, est appelé "action". La chaîne de caractères formée par la concaténation de l'étiquette portée par ce sommet et de l'étiquette portée par l'arc qui lui est incident extérieurement, exprime, pour l'utilisateur, l'exécution d'une assignation.

- Un sommet du graphe de degré intérieur 1 et de degré extérieur zéro ou 1, et dans ce cas son suivant est le titre de la table ou un titre de sous-table, est appelé rupture de séquence. L'arc qui lui est incident extérieurement, s'il existe, porte l'étiquette ✱.

5.1.2.2 - Remarques

Remarque 1

La définition des entrées étendues que nous proposons ne s'écarte pas dans son principe de la définition classiquement retenue, et décrite au chapitre 2. Une

condition reste formée de deux parties, l'une se trouve dans la souche, l'autre en entrée et forme l'entrée étendue.

Les tables de décisions pour être admises par les processeurs de tables doivent être exprimées en termes conventionnels, qui obéissent à une grammaire précise. Souches et entrées sont écrites en un langage qui s'approche du langage cible. Notre sujet n'impose pas cette limitation et tout texte qui exprime une condition ou une action peut être admis pour représenter cette condition ou cette action. Les entrées étendues pourront alors être les parties différentes de deux ou plusieurs textes.

Remarque 2

La définition d'une table à entrées étendues fait référence à un graphe de structure particulière. Elle se rapproche sensiblement de la définition que nous avons donnée, d'une table à entrées limitées. Elle s'en distingue par trois traits principaux :

- Les étiquettes portées par les sommets et les arêtes ne permettent plus de distinguer leurs natures : assignation ou conditions. Elles n'obéissent plus au formalisme classique :

$$c = p\ (L_1, \ldots, L_n)$$

$$L \leftarrow b\ (L_1, \ldots, L_m)$$

> La chaîne de caractères qui forme une étiquette n'est plus interprétable que par l'homme.

La conséquence en est que la vérification d'un programme à l'aide de telles tables documentaires sera plus délicate. Une erreur détectée pourra provenir aussi bien du programme source que des directives qui ont affectées les tables.

- La distinction entre sommets condition action et rupture de séquence ne peut plus provenir de l'interprétation déterministe des étiquettes portées par les sommets. C'est pourquoi

> La définition d'une table de décisions à entrées étendues fait porter la distinction sur le nombre de suivants et d'antécédents de chaque sommet.

Les noms des sous-tables (et éventuellement le titre de la table) sont les seuls sommets qui ont plusieurs antécédents.

Une rupture de séquence admet pour seul suivant éventuel le titre de la table ou le nom d'une sous-table titrée.

Une action a un seul suivant.

Enfin les sommets condition se distinguent par le fait qu'ils sont les seuls sommets qui ont plusieurs suivants. Ce nombre est maintenant variable d'une condition à l'autre et peut être supérieur à 2.

- Enfin nous avons introduit la singularité des tables à entrées étendues de présenter des circuits auxquels n'appartient pas le titre de la table.

5.1.2.3 - Termes usuels

Tout comme dans le cas des tables à entrées limitées, nous appellerons : Règle d'une table de décisions à entrées étendues, un chemin élémentaire dans le graphe de la table, dont l'extrémité initiale est le titre et l'extrémité finale une rupture de séquence.

La souche condition est l'ensemble des étiquettes portées par les sommets conditions du graphe. La souche action est définie à l'aide de l'ensemble des étiquettes portées par les sommets actions de la table. Le même algorithme que celui qui a été décrit en 3.3.2 permet de simplifier et d'ordonner cet ensemble.

Les entrées actions et conditions sont définies par les étiquettes portées par les arcs du graphe. Elles sont associées au sommet extrémité initiale de l'arc.

5.1.2.4 - Conclusion

La définition des tables de décisions à entrées étendues ne fait appel qu'à la structure du graphe. Le caractère général des textes de souches et d'entrées n'est pas défini par une axiomatique particulière, comme cela a été fait au chapitre 3. Les tables à entrées étendues seront construites à partir des tables à entrées limitées, par des transformations plus complexes que la simple recherche des parties communes à deux textes de souche. Pour obtenir une documentation qui s'écarte des intermédiaires de programmation et fasse abstraction des détails informatiques, aux niveaux de documentation les plus élevés, nous sommes amenés à accepter dans les souches, toute phrase du langage courant qui sera jugée, par l'utilisateur et indépendamment de règles syntaxiques, comme caractérisant ou un état ou un traitement, représenté dans les tables à entrées limitées par une ou plusieurs lignes de souche.

5.1.3 - Forme de codage des tables de décisions à entrées étendues

Après avoir défini le concept de tables de décisions à entrées étendues et avant de présenter le traitement et la manipulation de ces tables, nous décrirons le mode de représentation interne qui a été choisi.

Pour des raisons évidentes d'homogénéité, la forme de codage retenue sera semblable à celle qui a été créée pour les tables à entrées limitées. L'information nécessaire à l'édition sera contenue dans quatre fichiers :

1) Fichier des noms de tables et de sous-tables

2) Fichier des souches.

3) Fichier des entrées. Ce fichier n'a pas été défini pour les tables à entrées limitées. Chaque enregistrement contient les entrées différentes attachées à un même texte de souche.

4) Fichier du codage qui, ici encore, est une suite continue de clés d'accès aux fichiers précédents et représente un codage du graphe de la table de décisions.

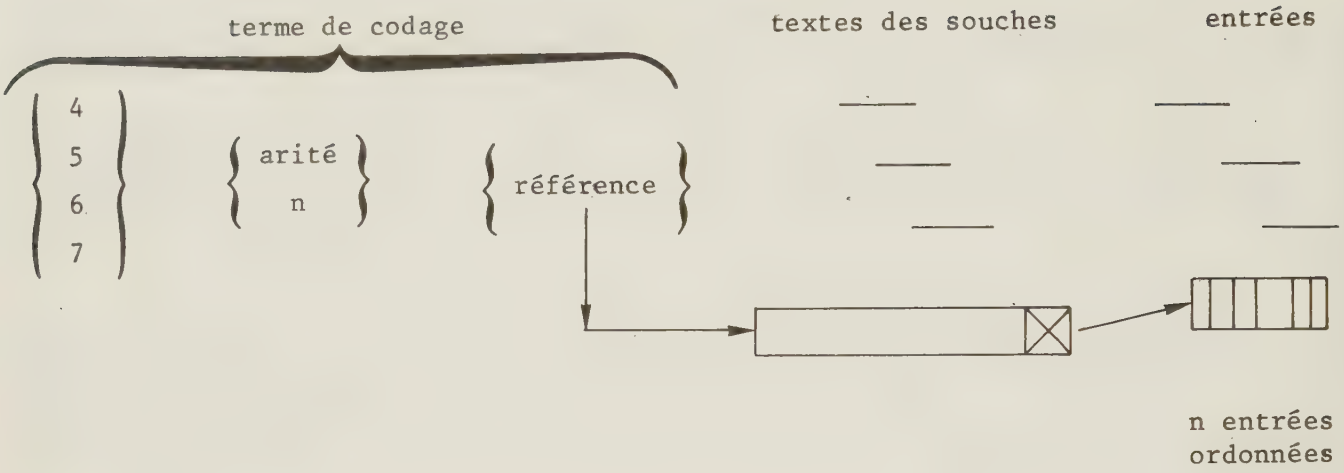
5.1.3.1 - Présentation

La forme de représentation en mémoire d'ordinateur des tables de décisions à entrées étendues, étant construite de façon analogue à celle des tables à entrées limitées, nous nous attacherons à décrire les différences. Elles tiennent à la représentation des :

- titre de sous-tables titrées. Le pointeur 1 non encore utilisé est caractéristique d'un titre de sous-table. La référence associée dans le terme de codage est l'adresse du nom attaché à la sous-table.

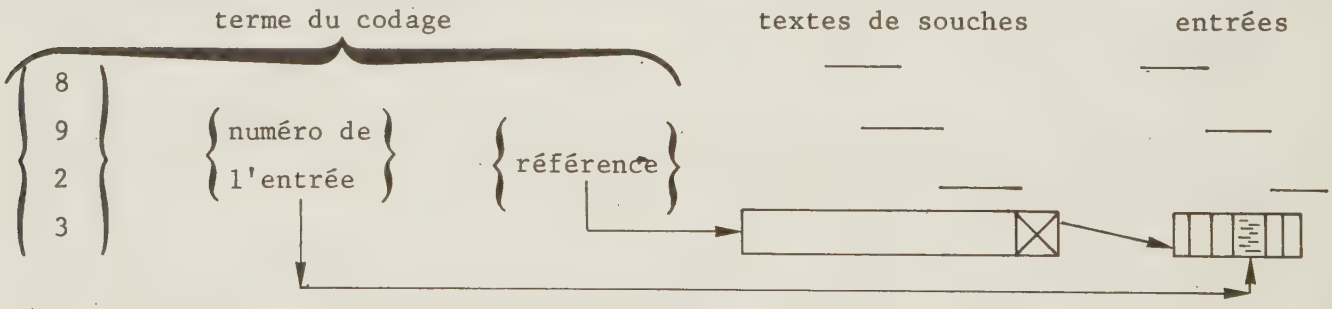
- degrés extérieurs des conditions. A un sommet condition est associé dans la forme préfixée de codage un "opérateur" d'arité égale au degré extérieur du sommet. Les caractères 4, 5, 6, 7 sont caractéristiques d'un sommet condition. L'arité est ensuite indiquée dans le terme par deux chiffres. La référence qui suit est celle du texte de souche de la condition, chaîné à la suite ordonnée des étiquettes (entrées) portées par les arcs incidents extérieurement au sommet.

Le schéma suivant décrit l'enchaînement :



- entrées actions étendues. A un sommet action ou rupture de séquence du graphe est associé un terme formé d'un pointeur, respectivement 8, 9 et 2, 3 ; puis du numéro de l'entrée correspondant à l'arc incident extérieurement, enfin de la référence du texte de la souche action.

Le schéma suivant résume les enchaînements d'accès aux textes et entrées :



5.1.3.2 - Forme normale de Bacchus

Le vocabulaire terminal est formé de :

- l'ensemble $\mathcal{C}$ des clés d'accès C à la souche condition,
- l'ensemble $\mathcal{A}$ des clés d'accès A à la souche action,
- l'ensemble $\mathcal{S}$ des clés d'accès S aux ruptures de séquence,

- l'ensemble $\mathcal{P}$ des clés d'accès P aux titres de tables et sous-tables,
- l'ensemble N des nombres entiers n positifs, inférieurs à 100,
- la parenthèse 0000,
- les chiffres 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

L'ensemble formé :

- d'un pointeur d'un chiffre,
- d'un nombre de deux chiffres,
- d'une référence A, C, S ou P, 4 chiffres sera appelé terme de codage. Il lui correspond un enregistrement logique dans les fichiers qui mémorisent le codage d'une table.

Un terme de codage d'une table à entrées étendues contient donc un élément de plus qu'un terme de codage de table à entrées limitées ; à savoir le nombre de 2 chiffres qui, pour une condition, exprime le degré extérieur du sommet et pour une action le numéro de l'entrée.

Les règles de formation sont :

L'axiome est TDEE.

R1 TDEE ::= 0 00 0000 0 00 P 0 00 0000 <TD> 0 00 0000

R2 <TD> ::= <opérande> | 7 n C <opérande> ... n fois ... <opérande>

R3 <opérande> ::= <TD> | 1 00 P <TD>

R4 <opérande> ::= 2 01 S | 8 n A <séquence> |
5 n C <opérande> ... n fois ... <opérande>

R5 <séquence> ::= 3 01 S | 9 n A <séquence> |
6 n C <opérande> ... n fois ... <opérande> |
4 n C <opérande> ... n fois ... <opérande> |
1 00 P <séquence>

On voit apparaître dans ces règles les deux différences qui distinguent le codage des tables à entrées étendues.

- L'intervention du pointeur 1 dans les règles 3 et 5. Il permet de caractériser le nom d'une sous-table.

- La présence de n facteurs opérandes consécutifs à la suite d'une condition n est précisé dans le terme de la condition.

5.2 - CONSTRUCTION DES TABLES DE DECISIONS A ENTREES ENTENDUES

Après avoir défini les tables de décisions à entrées étendues, et choisi une représentation interne, la deuxième partie de ce chapitre sera consacrée au traitement de ces tables.

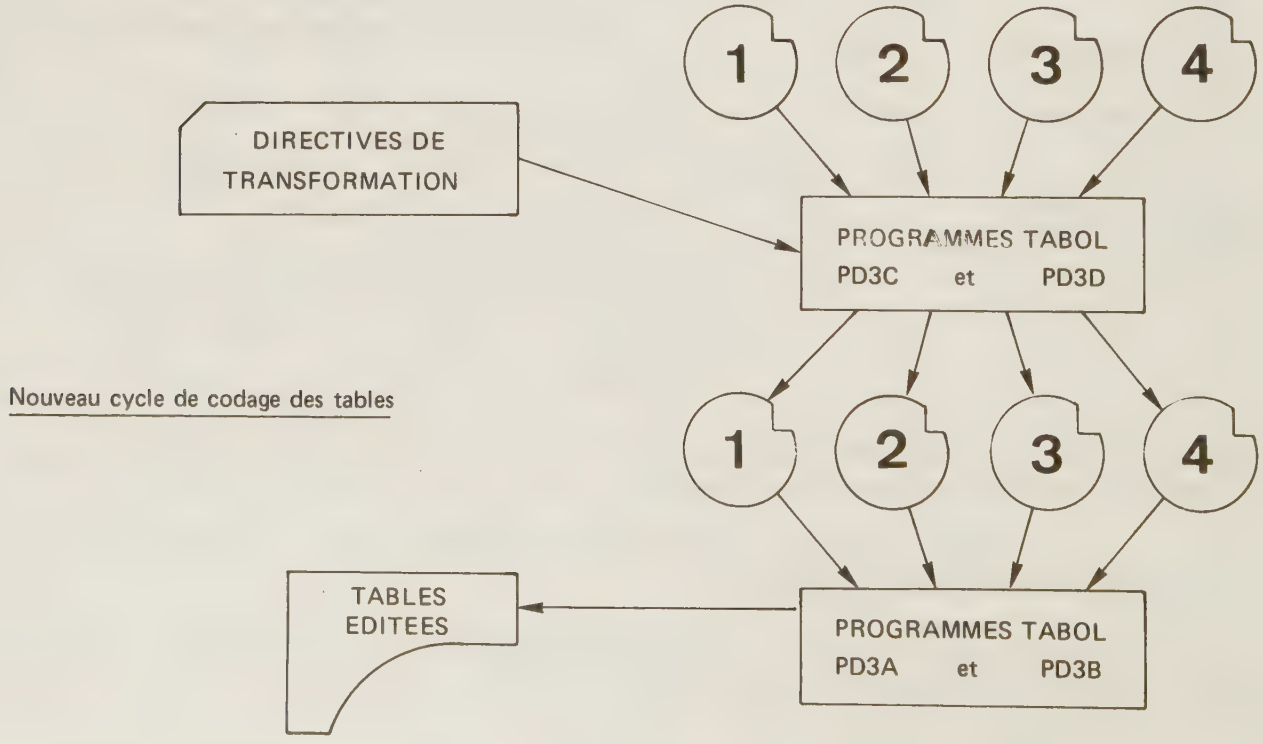
La construction des tables à entrées étendues est réalisée à partir :

- d'une part des fichiers représentant le dernier cycle des tables éditées. Il s'agit des fichiers décrits dans la première partie et qui contiennent :

  1) le nom des tables et sous-tables titrées,
  2) les textes de souches,
  3) les textes d'entrées,
  4) les phrases de codage des tables :

- d'autre part d'un fichier de directives de transformations fourni par l'utilisateur.

La construction et le traitement des tables à entrées étendues peut donc s'interpréter comme des mises à jour successives des quatre fichiers principaux. Le schéma suivant résume le fonctionnement :



Le chapitre 7 et l'annexe II qui présentent un exemple d'utilisation permettront de détailler la nature, la syntaxe et l'emploi de chacune des 8 directives disponibles. Le chapitre 8 décrit les programmes de la chaîne TABOL et en particulier les deux programmes PD3C et PD3D qui traitent les tables à entrées étendues. Le lecteur y trouvera une présentation rapide de la pratique des algorithmes de traitement des directives de transformations.

Après une présentation rapide de la directive qui autorise l'introduction de textes dans la souche action et d'entrées actions étendues, nous nous intéresserons, ici, au traitement de la directive principale qui permet la transformation de la souche et des entrées conditions d'une table.

Les autres directives donnent lieu à des traitements beaucoup plus élémentaires qui ne feront pas l'objet ici d'une présentation théorique.

5.2.1 - Entrées actions étendues

Les études qui ont eu pour objet l'analyse et l'emploi des tables de décisions à entrées étendues se sont limitées généralement, à l'étude des entrées condition de la table. L'introduction dans les tables d'entrées actions étendues permet de remplir deux objectifs différents :

- Rassembler sur une même ligne de la table des actions qui sont exécutées dans autant de règles différentes et ont des opérandes en commun, en général variables réceptrices ou expressions émettrices.

- Rassembler sur une même ligne de la table des actions exécutées en séquence, le long d'une règle dans la table. Le texte de souche est un libellé qui résume le traitement entrepris lors de l'exécution séquentielle des actions regroupées.

Les deux rôles ne sont pas exclusifs. L'indication des actions à regrouper, du texte de souche qui les résume et des règles et entrées étendues correspondantes est fournie par l'utilisateur. Le traitement auquel conduit cette directive consiste à effacer du codage les termes qui codent les actions à regrouper et des fichiers les textes correspondants, pour les remplacer par la seule occurence de l'action regroupée. Nous ne détaillerons pas davantage ici ce traitement particulièrement simple.

5.2.2 - Entrées conditions étendues

La directive de transformation des conditions d'une table a semblé suffisamment originale pour justifier la présentation théorique qui en est faite dans les pages suivantes.

L'exposé commence par une description intuitive du rôle que remplit la directive. Nous nous appuierons sur un exemple (*5.2.2.1*).

Dans le paragraphe suivant est posée la définition de conditions sémantiquement liées, c'est-à-dire susceptibles de faire l'objet de la directive (*5.2.2.2*).

Nous établirons ensuite une condition nécessaire et suffisante, équivalente à la définition précédente et qui permet de traduire sur le graphe de la table, les caractéristiques des conditions sémantiquement liées (*5.2.2.4*).

Enfin le dernier paragraphe décrit le traitement du codage de la table qui permet de rendre effectives les transformations demandées dans la directive (*5.2.2.5*).

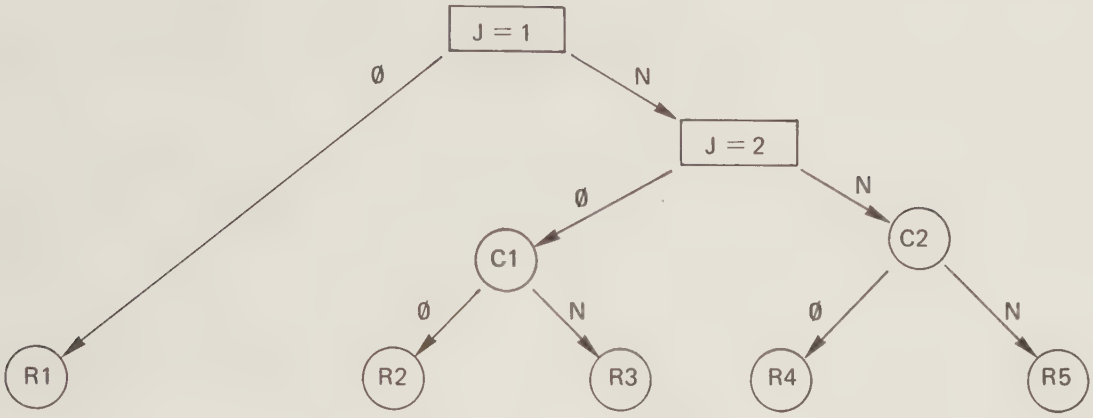
Le traitement, décrit théoriquement ici, donnera lieu, au chapitre 8, à l'écriture de l'algorithme pratique d'exécution d'une directive de conditions.

5.2.2.1 - Présentation

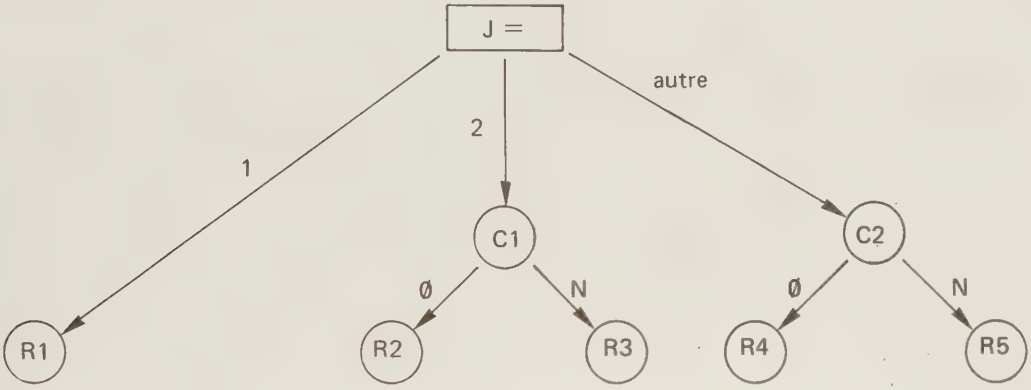
Pour construire des entrées conditions étendues, la présentation classique propose d'étudier les parties communes à plusieurs textes de souche de façon à étiqueter un sommet par la partie commune et les arcs par les parties singulières. Le résultat est un graphe dont un sommet admet plusieurs suivants. Rappelons qu'il a paru nécessaire pour notre objet, que cette transformation du graphe puisse être contrôlée par l'utilisateur, qui fournit :

- l'indication des sommets qu'il entend voir regrouper,
- le texte de la partie commune, nouvelle étiquette du sommet,
- et celui des entrées ou étiquettes des arcs.

Ainsi le graphe de l'exemple suivant :



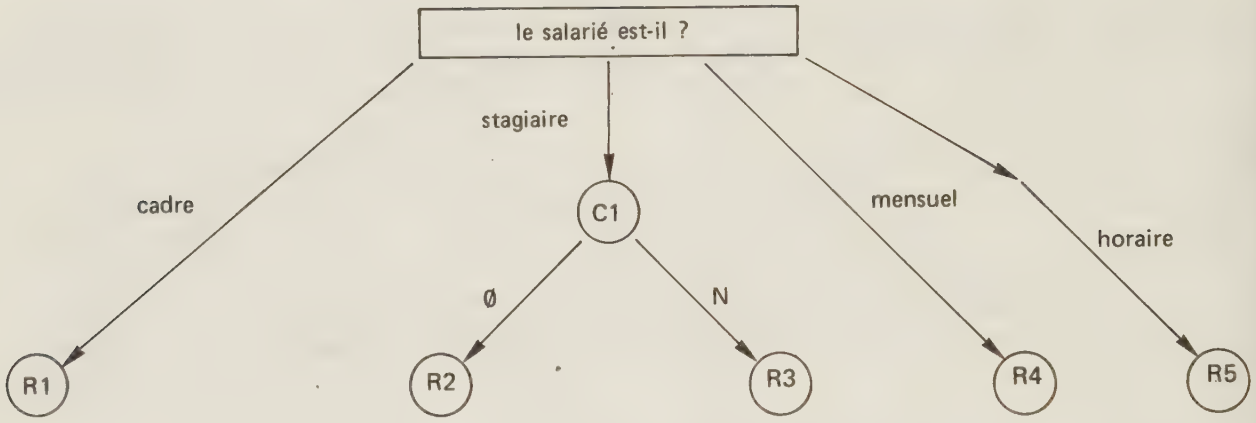
peut être remplacé par le suivant :



Les deux conditions regroupées J = 1 et J = 2 sont sémantiquement liées puisqu'elles s'intéressent à la valeur de la même location. La condition C2 dont le texte n'est pas précisé, et qui partage les cas où J ne vaut ni 1 ni 2, caractérise des cas d'espèces liés à la valeur de J. Si le texte de C2 était J = 3, il eût été possible de regrouper cette condition aux précédentes. Nous chercherons à autoriser le regroupement de C2 aux conditions J = 1 et J = 2 alors même que le texte de C2 n'a aucun élément commun avec elles.

Nous prendrons l'exemple qui à la valeur 1 de J associe les cadres d'une entreprise, à la valeur 2 les ouvriers non titulaires ou stagiaires, aux autres valeurs les ouvriers titulaires. La condition C2 sépare les mensuels des horaires. Nous dirons que les conditions J = 1, J = 2 et C2 sont sémantiquement liées aux valeurs de ces conditions.

Un graphe à entrées étendues équivalent au regroupement de ces trois conditions pourrait être :



Il apparaît que la condition C1 est sémantiquement liée aux précédentes. Au contraire, il est impossible de regrouper les conditions C2 et C1 seules ; la connaissance de leurs valeurs ne permet pas de séparer les règles 2 à 5.

Le résultat présenté par la figure ci-dessus, du regroupement des 3 conditions est un graphe de tables de décisions à entrées étendues. Le sommet est étiqueté par la partie commune du nouveau texte de condition. Le regroupement arbitraire de lignes conditions ayant des parties communes sur la forme éditée de la table, est remplacé par le regroupement de sommets liés par la structure du graphe. L'avantage d'une telle méthodologie est de permettre à l'édition la séparation de cas d'espèces à l'aide d'une seule ligne condition, quelle que soit la nature ou la complexité des conditions réellement programmées pour distinguer les cas,

C'est le propre de la partie condition d'une table de décisions que de permettre de séparer tous les cas. Il est donc possible de regrouper sur une seule ligne toutes les conditions d'une table. Il est douteux qu'une telle solution entraîne une documentation efficace. Et finalement seul l'utilisateur pourra déterminer les lignes conditions qu'il juge sémantiquement liées.

Nous avons montré que le regroupement de conditions n'est envisageable que pour des sommets liées par la structure du graphe de la table. Par extension, nous dirons que de tels sommets sont sémantiquement liés pour exprimer que les conditions qui les étiquettent peuvent être regroupées. Les valeurs des conditions séparent des cas (ou des règles) que nous dirons complémentaires.

5.2.2.2 - Conditions sémantiquement liées

Ce paragraphe précise, en les définissant théoriquement, les notions de lien sémantique et de règles complémentaires, introduites de façon intuitive au paragraphe précédent.

La forme éditée d'une table de décisions fait disparaître la structure du graphe qui la code. Nous définirons donc la notion de lien sémantique par référence à la forme de la table. Le problème du regroupement de conditions ne se pose que pour les conditions d'une même sous-table distinguée à l'édition.

Nous appellerons jeu de conditions un sous-ensemble des conditions d'une même sous-table d'édition.

Soit n le nombre d'entrées indifférentes d'une règle sur les lignes d'un jeu de conditions $\mathcal{C}$. Nous appellerons poids de la règle relativement au jeu de conditions le nombre 2^n.

Nous dirons qu'un ensemble de règles est sans ambiguïté relativement à un jeu de conditions si, quelle que soit la valeur affectée à chaque condition du jeu, une seule règle est satisfaite. Rappelons qu'il faut et suffit que les entrées de deux règles quelconques diffèrent par au moins une entrée distincte non indifférente sur la même ligne.

Nous dirons que des règles sont complémentaires relativement à un jeu de conditions si et seulement si :

- elles sont sans ambiguïté sur le jeu,
- la somme de leur poids est 2^m, si m est le nombre de conditions du jeu,
- pour toute condition du jeu, il existe une règle au moins de l'ensemble dont l'entrée sur la condition n'est pas indifférente.

L'énoncé suivant est équivalent aux deux premières hypothèses :

- quelle que soit la valeur affectée à chaque condition du jeu une règle et une seule est satisfaite.

La troisième hypothèse signifie qu'une condition du jeu ne peut être indifférente pour toutes les règles. La condition de non ambiguïté impose alors que toute condition sépare au moins deux règles par des entrées non indifférentes.

Définition

Un jeu de conditions est sémantiquement lié si et seulement si il existe un ensemble de règles complémentaires relativement au jeu.

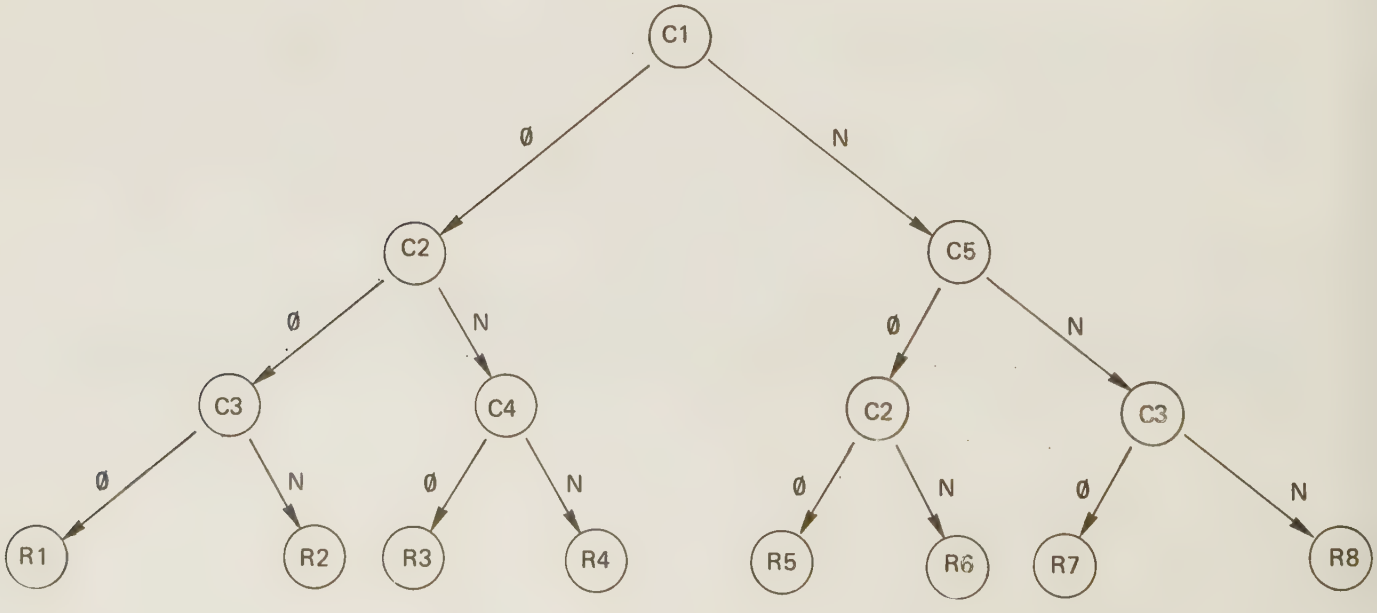
Rappelons que nous imposons par définition, que seules des conditions sémantiquement liées pourront être regroupées par une seule ligne de la souche conditions. Il résulte de la définition que la directive de regroupement devra fournir autant d'entrées différentes qu'il y a de règles dans un ensemble de règles complémentaires.

Un ensemble de règles complémentaires n'est pas unique. L'exemple de la figure suivante le montre : les règles 1, 2 et 5 sont complémentaires, mais aussi les règles 1, 3 et 6 pour le jeu (C2, C3).

	R1	R2	R3	R4	R5	R6	R7
C1	Ø	Ø	Ø	Ø	Ø	Ø	N
C2	Ø	N	N	N	N	N	
C3		Ø	Ø	Ø	N	N	
C4		Ø	N	N			
C5			Ø	N	Ø	N	

5.2.2.3 - Sous-tables parallèles

Considérons l'exemple du graphe suivant :



Il lui correspond la forme éditée suivante :

	R1	R2	R3	R4	R5	R6	R7	R8
C1	Ø	Ø	Ø	Ø	N	N	N	N
C2	Ø	Ø	N	N	Ø	N		
C3	Ø	N					Ø	N
C4			Ø	N				
C5					Ø	Ø	N	N

Figure *5.2.2.3.1*

Il apparaît que les conditions C2 et C3 ont chacune deux occurences dans le graphe. La forme éditée fait apparaître ces deux occurences. Les lignes C2 et C3 dans la sous-table définie par la valeur Ø de C1 peuvent être regroupées. Un ensemble de règles complémentaires peut être R1, R2, R3. Au contraire, elles ne peuvent l'être dans la sous-table définie par la valeur N de C1 ; le poids total des règles 5, 6, 7, 8 est $8 > 2^2$.

Les conditions C2 et C3 de la figure suivante sont sémantiquement liées dans les deux sous-tables parallèles séparées par la valeur de C1. Cependant dans la première le degré extérieur de la condition regroupée sera 3 et dans la seconde il sera 4. Les règles complémentaires sont respectivement : R1, R2, R3 et R4, R5, R6, R7. La directive de regroupement devra préciser les deux ensembles de règles complémentaires.

	R1	R2	R3	R4	R5	R6	R7
C1	Ø	Ø	Ø	N	N	N	N
C2	Ø	N	N	Ø	Ø	N	N
C3		Ø	N	Ø	N	Ø	N

Figure *5.2.2.3.2*

Pour ne pas alourdir le texte des démonstrations qui suivent, nous nous limiterons au cas où toute condition du jeu n'a qu'une seule occurence dans la sous-table d'édition. Les résultats s'étendent de façon simple au cas d'occurences multiples en limitant les énoncés à chacune des sous-tables parallèles qui ne contiennent qu'une seule occurence de chaque condition du jeu.

Cette restriction permet d'annoncer maintenant le résultat suivant : tous les ensembles de règles complémentaires, qui chacun définissent la liaison sémantique de n conditions ont le même nombre d'éléments. Ce nombre sera le degré extérieur du sommet étiqueté par le condition regroupée.

La notion de lien sémantique entre des conditions est intuitive. Nous avons construit un critère simple pour reconnaître un tel lien souvent implicite voire subjectif. Il reste que la directive de regroupement de conditions est d'un emploi délicat si le jeu de conditions a plusieurs occurences dans différentes sous-tables parallèles.

5.2.2.4 - La directive de regroupement des conditions

Le paragraphe 6.2 du chapitre 7 détaille la forme syntaxique dans laquelle doivent être fournies les indications nécessaires au regroupement de conditions :

- le jeu de conditions à regrouper,
- le texte de souche résumé,
- l'ensemble des règles complémentaires,
- les entrées étendues.

5.2.2.4.1 - Condition nécessaire et suffisante de liaison sémantique

Pour construire le nouveau graphe qui regroupe les sommets conditions indiqués dans le jeu, nous chercherons une sous-table telle que :

a) Son titre V soit étiqueté par une condition du jeu.

b) Les règles de l'ensemble complémentaire soient des règles de la sous-table de titre V.

c) Chaque condition du jeu soit l'étiquette d'un sommet de la sous-table.

L'existence d'une telle sous-table, quel que soit le graphe associé à la forme éditée de la table de décisions, est une condition nécessaire et suffisante pour qu'un jeu de conditions soit sémantiquement lié.

Cette propriété caractéristique permet la traduction en termes de graphe de la définition d'une liaison sémantique de conditions, exprimée en termes de forme éditée. Elle permettra donc de réaliser le traitement qui transforme le graphe de la table, en accord avec une directive écrite d'après la forme éditée.

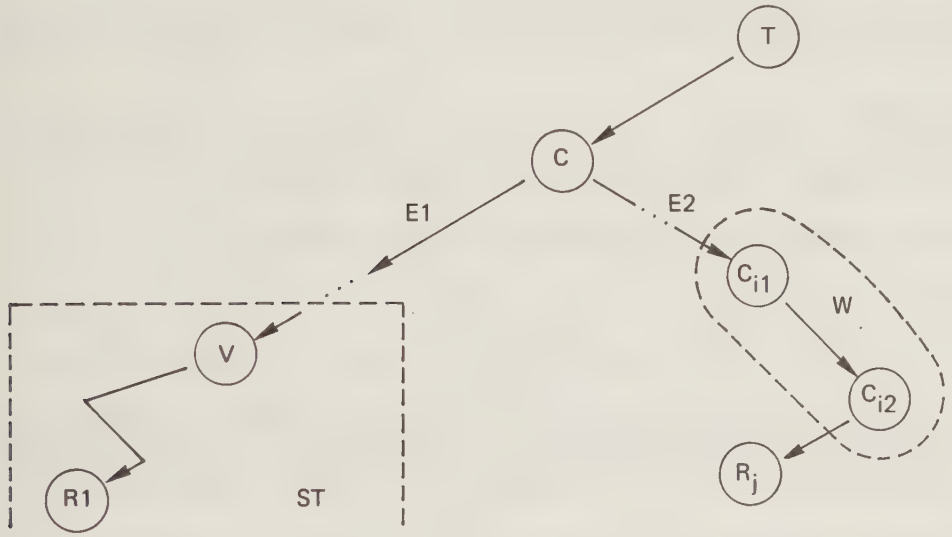
5.2.2.4.2 - Condition nécessaire

Démontrons ce résultat. Soient $\mathcal{C}$ un jeu de conditions sémantiquement liées d'une sous-table d'édition et $\mathcal{E}$ un ensemble de règles complémentaires pour $\mathcal{C}$.

R1 est une règle de $\mathcal{E}$, c'est-à-dire un chemin dans le graphe de la sous-table. Soit V la première occurence d'une condition du jeu sur R1. Montrons que la sous-table ST de titre V vérifie les conditions cherchées.

a) Par construction son titre appartient au jeu $\mathcal{E}$.

b) Montrons par l'absurde que toutes les règles de $\mathcal{E}$ sont des règles de ST. Soit T le titre de la sous-table entière. Si R_j, règle de $\mathcal{E}$, n'est pas une règle de ST les chemins T-R1 et T-R_j se séparent en C aïeul de V.



Soit alors W l'ensemble des conditions du jeu $\mathcal{E}$ qui ont une occurence sur le chemin T-Rj. Cet ensemble n'est pas vide, c'est une conséquence de la troisième hypothèse fait dans la définition d'un ensemble de règles complémentaires.

Une condition de W a encore au moins une occurence dans ST. S'il n'en était pas ainsi, R1 et R_j seraient ambiguës pour le jeu de conditions puisque distinguées par des conditions toutes différentes. Ce qui est contraire à l'hypothèse de définition d'un ensemble de règles complémentaires.

Une condition de W a alors deux occurences dans la sous-table considérée, ce qui contredit l'hypothèse que nous avons posée. Il en résulte que toutes les règles de l'ensemble complémentaire $\mathcal{E}$ sont des règles de ST.

c) La troisième hypothèse qui a servi à définir un ensemble de règles complémentaires, impose que toute condition du jeu serve à séparer effectivement deux règles. Toutes les règles appartenant à ST, chaque condition du jeu a au moins une occurence dans ST.

La sous-table de titre V est donc bien la sous-table cherchée.

5,2.2.4,3 - Réciproque

Soient, maintenant, un jeu de m conditions et un graphe de la table de décisions. Par hypothèse, nous supposons l'existence d'une sous-table dont le titre est une condition, soit Co, du jeu et dans laquelle toutes les conditions du jeu ont une occurence,

Construisons un ensemble de règles dont nous montrerons qu'elles sont complémentaires relativement au jeu.

1) Soit C = Co.

2) *a* Soit C1 le premier suivant de C non encore étudié.

b Si C1 n'appartient pas au jeu faire C ← C1 aller en 2 *a*.

3) *a* Si la sous-table de titre C1 contient une occurence de toutes les conditions du jeu faire Co ← C1 et recommencer l'algorithme en 1.

b Si la sous-table de titre C1 ne contient plus aucune occurence des conditions du jeu ajouter une règle quelconque, prise dans chaque branche descendant de C1 à l'ensemble de règles. Aller en 4).

c Si la sous-table de titre C1 ne satisfait ni *a* ni *b* faire C ← C1. Reprendre en 2.

4) Si tous les suivants de C ont été étudiés, prendre pour C l'antécédent de C. Dans les deux cas aller en 2). S'il n'ya plus d'antécédent, fin d'algorithme.

Les règles ainsi construites sont sans ambiguïté relativement au jeu. Le choix fait en 3)*b* et le cheminement dans le graphe, imposent que deux règles quelconques sont choisies dans deux branches différentes d'une même condition du jeu.

Les règles forment un ensemble complet puisque pour toute condition du jeu une règle au moins est choisie dans chaque branche descendante. C'est encore une conséquence du cheminement entrepris.

Enfin l'algorithme permet de décrire tous les sommets du graphe et en particulier tous ceux qui sont étiquetés par une condition du jeu. Toute condition du jeu intervient donc dans la séparation de deux règles.

L'ensemble des règles ainsi construit est complémentaire et le jeu de conditions est sémantiquement lié.

5.2.2.5 - Codage normal

La directive de regroupement des conditions conduit à modifier la sous-table caractéristique dont nous venons de montrer l'existence.

Il s'agit de rechercher en premier lieu la racine U_j de la branche de la sous-table qui contient la règle j de l'ensemble complémentaire. Soit S_j la rupture de séquence qui définit la règle j. U_j est un aïeul de S_j tel que :

- Aucune condition du jeu n'étiquette un sommet descendant de U_j.

- L'antécédent de U_j est étiqueté par une condition du jeu.

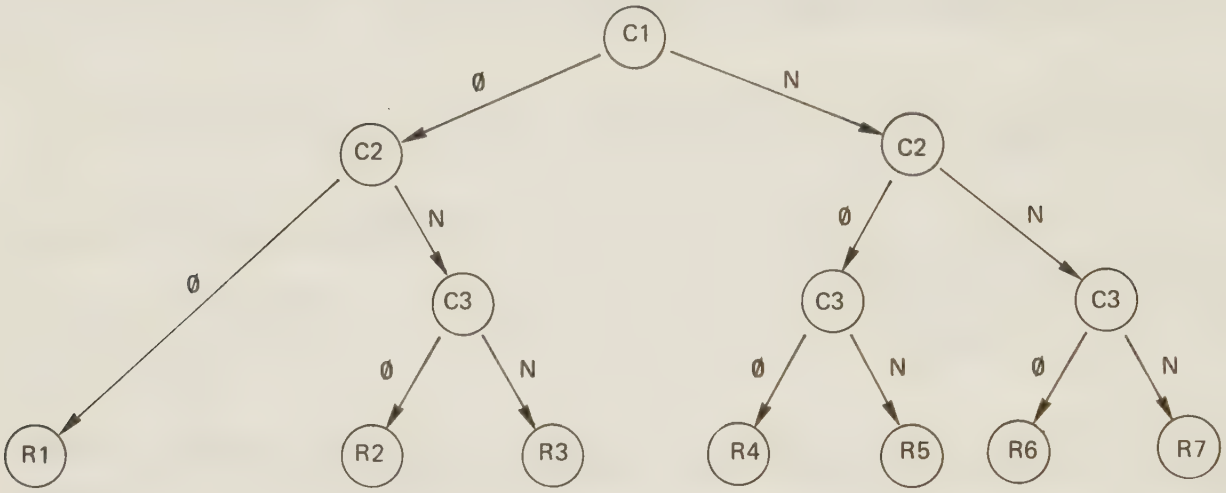
Quelles que soient les valeurs affectées aux conditions, une règle de l'ensemble complémentaire au moins est satisfaite. Il en résulte que sur toute branche descendant d'une condition du jeu dans la sous-table caractéristique, et donc sur le chemin de toutes les règles de la sous-table il existe une racine U_j.

Nous appellerons arborescence caractéristique de la liaison sémantique le sous-graphe de la sous-table caractéristique dont la racine est le titre de la sous-table et les sommets pendants les sommets U_j.

L'arborescence caractéristique et le codage qui lui correspond sont dits normaux pour le jeu de conditions si tous les sommets non pendants sont étiquetés par une condition du jeu.

Pour illustrer cette notion nous reprendrons l'exemple de la figure

5.2.2.3.2. Le graphe qui correspond à cette forme éditée est le suivant :



Le jeu de conditions est formé de C2 et C3. Il existe deux ensembles de règles complémentaires pour ce jeu, correspondant à deux sous-tables parallèles, séparées par la valeur de C1.

1) R1 R2 R3

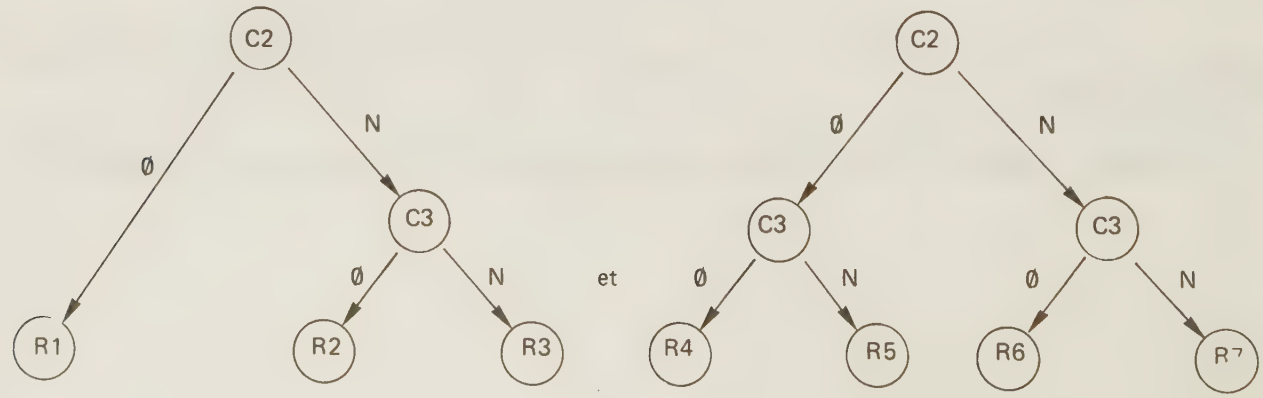
2) R4 R5 R6 R7

En l'absence d'une souche action, les racines U_j des branches pendantes sont ici schématisées par les règles elles-mêmes.

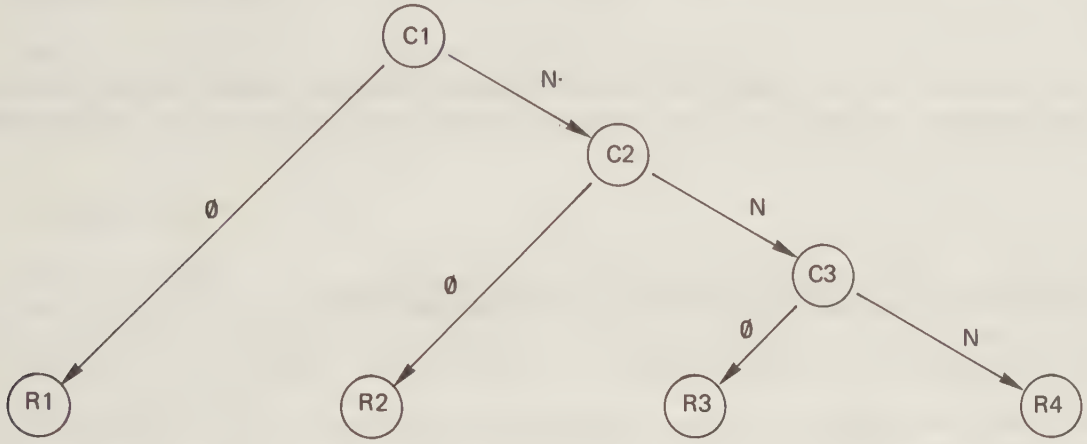
1) U1 = R1, U2 = R2, U3 = R3

2) U1 = R4, U2 = R5, U3 = R6, U4 = R7

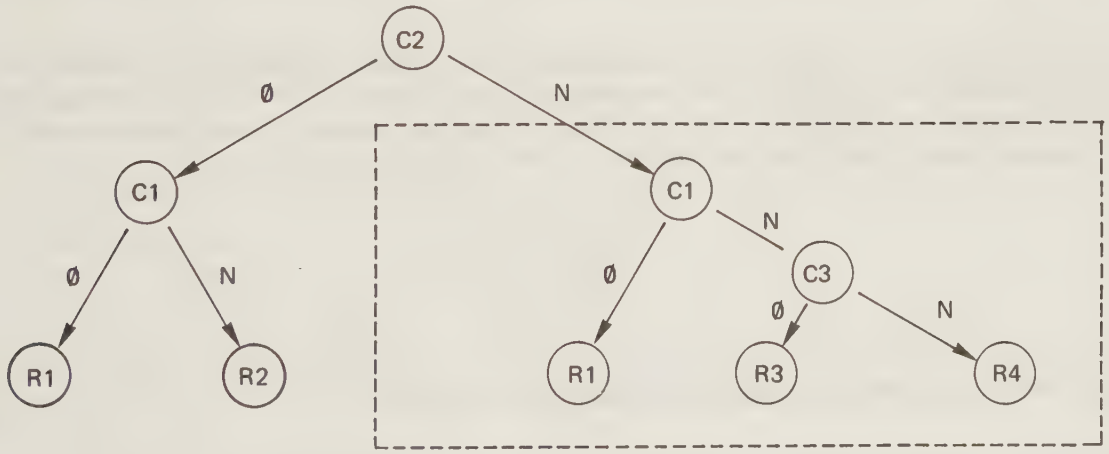
Les deux racines des arborescences caractéristiques des liaisons sémantiques sont les deux sommets étiquetés par C2. Et les deux arborescences caractéristiques sont :



Ces deux sous-graphes sont normaux pour C2 et C3. Il n'y intervient aucune autre condition. L'exemple suivant montre qu'un jeu de conditions peut être sémantiquement lié et le graphe correspondant n'être pas normal.



Considérons le jeu de conditions (C1, C3) et l'ensemble des règles : R1, R3, et R4. Ces règles sont complémentaires relativement au jeu. Celui-ci est donc sémantiquement lié. Le graphe dessiné est le graphe caractéristique correspondant. La présence d'un sommet étiqueté par C2 interdit à ce graphe d'être normal. Un graphe équivalent quant à la logique de la table peut être construit en étiquetant la racine par la condition hors du jeu :



Le graphe caractéristique de la liaison sémantique est encadré. Il est normal.

Le nombre de sommets d'un sous-graphe caractéristique d'une liaison sémantique est fini ; le nombre de conditions hors jeu l'est aussi, et un nombre fini de transformations du graphe permettent de construire un graphe normal.

Le traitement de la directive de regroupement consiste alors à supprimer les sommets non pendants du graphe normal, caractéristique de la liaison, pour les remplacer par un unique sommet étiqueté par le texte de souche précisé dans la directive.

Les arêtes qui joignent ce nouveau sommet aux racines U_j reçoivent chacune une étiquette, entrée de la condition regroupée sur les règles qui descendent de U_j.

5.3 - EDITION DES TABLES DE DECISIONS A ENTREES ETENDUES

Les tables à entrées étendues, construites par mise à jour des fichiers qui contiennent le codage du cycle précédent, doivent ensuite être éditées sur imprimante.

Ce sont les même programmes de la chaîne, PD3A et PD3B, et donc les mêmes algorithmes qui réalisent l'édition des tables de décisions à entrées limitées et à entrées étendues. Pour obtenir ce résultat les phrases de codage des tables à entrées limitées sont harmonisées avec la syntaxe qui permet de décrire des entrées étendues.

Chaque terme de codage est transformé en un terme complet contenant le nombre de 2 chiffres qui précise les entrées. A une instruction conditionnnelle COBOL sont attachées les deux entrées Ø et N qui ne seront plus distinguées d'entrées étendues générales. Le terme de la phrase de code devient :

$$\left\{\begin{matrix}4\\5\\6\\7\end{matrix}\right\} \qquad \underline{\underline{0\ 2}} \qquad C$$

pour signifier qu'une telle condition a deux suivants.

De même à une assignation ou une rupture de séquence est associée l'entrée ✱, le numéro de l'entrée indiquée dans le terme sera 01.

L'édition des tables à entrées limitées, étendues ou mixtes ne seront plus distinguées. Nous avons signalé dans la dernière partir du chapitre 4 l'essentiel de l'algorithme d'édition. Les modifications qu'il faut apporter à ces algorithmes

tiennent à la représentation des tables de décisions à entrées étendues :

Souche condition

Deux instructions conditionnnelles sont déclarées identiques non seulement si le texte de souche est identique, mais encore si les entrées le sont et ont le même ordre.

Souche action

Deux actions A_i et A_j dont le texte de souche est identique sont déclarées identiques. Si de plus aucune relation d'ordre ne les lie (c'est-à-dire si $q_{ij} = q_{ji} = 0$), une seule référence leur est associée suivant la règle décrite au chapitre 4. Les entrées différentes sont regroupées et attachées à la nouvelle action de la table.

Entrées conditions

Chacun des n opérandes d'une condition (la valeur de n est indiquée dans le terme qui référence la condition), doit être séparé. A l'opérande i est attaché la i ième entrée de la condition ; les règles dont la rupture de séquence se trouve dans cet opérande, contiennent donc l'entrée i sur la ligne de la condition. Suivant le schéma :

7 03 C R1 R2 | R3 R4 | R5

1° opérande 2° opérande 3° opérande

- La première entrée de C se trouvera sur les règles 1 et 2,
- la deuxième sur les règles 3 et 4,
- la troisième sur la règle 5.

Rappelons que les opérandes d'une condition sont séparées à l'aide de la ligne qui contient la position du dernier suivant de la sous-table induite par chaque terme.

Entrées actions

Pour chaque terme de référence à une action le nombre de deux chiffres qu'il contient indique le numéro de l'entrée qui est affecté. Les règles dont la rupture de séquence se trouve dans la sous-table induite par la référence à l'action, posséderont cette entrée sur la ligne de l'action. Ainsi :

9 03 A ... R1 R2 R3 ... 9 02 A ... R4

conduira à une ligne A formée :

- de l'entrée numéro 3 sur les règles 1, 2 et 3
- et de l'entrée numéro 2 sur la règle 4.

Le jeu de tables ainsi édité pourra à son tour faire l'objet de transformations au moyen de nouvelles directives, pour créer un nouveau cycle de tables. Sera ainsi présenté un nouveau document plus synthétique pour faire apparaître plus clairement la hiérarchie des modules fonctionnels.

CHAPITRE 6

HOMOLOGATION DES PROGRAMMES DE GESTION A L'AIDE DE TABLES DE DÉCISIONS

Une documentation claire et efficace obtenue, le problème demeure du contrôle de la validité du programme. Nous emprunterons au Centre d'Electronique de l'Armement les définitions suivantes.

L'homologation d'un produit software est un contrôle général indépendant d'une utilisation particulière, dont l'objectif est de s'assurer que le programme est conforme à des spécifications générales, qu'il est utilisable dans les conditions normales d'un centre de traitement, qu'il est bien documenté, etc. L'homologation, qui peut ne pas s'intéresser aux performances (de temps, de place) d'une implantation particulière, réalise une validation complète d'un programme d'usage général avant toute utilisation précise.

Nous distinguerons l'homologation de la recette, validation par le client d'une application particulière et dont l'objectif est le contrôle du programme dans le cadre d'un cahier des charges établi par l'utilisateur lui-même. La mesure des performances peut prendre ici un caractère essentiel.

Nous distinguerons encore l'homologation, du premier débourrage d'un module du programme. L'opération de débourrage fait partie intégrante de l'ensemble analyse-programmation. Si les moyens mis en oeuvre (jeux d'essais documentation) dans les deux cas sont souvent proches, les deux opérations se distinguent par leurs objectifs respectifs. Tandis que l'une contrôle la qualité d'un produit fini, la seconde est une étape particulière de la création.

6.1 - PRESENTATION

Nous détaillons dans ce chapitre quelques critères de contrôle de la qualité de la programmation proposée, en réalisant par là un début d'homologation.

Un contrôle efficace, quelqu'il soit, proviendra de la comparaison du programme donné, ou d'une représentation équivalente, à un algorithme abstrait de référence dont seul l'homme peut avoir la connaissance. L'intervention nécessaire de l'homme dans le contrôle accroît la difficulté qu'il y a de définir une démarche à la fois systématique et complète.

Le programme, les tables de décisions, l'organigramme sont des représentations de l'absolu de référence, au même titre que le cahier des charges le dossier d'analyse ou un jeu d'essais complet et ses résultats. Toutes ces représentations concrètes sont susceptibles d'être entachées d'erreurs et ne peuvent servir de terme de comparaison. Une erreur d'analyse se retrouvant dans la programmation, ne sera évidemment pas détectée par la comparaison du dossier d'analyse et du programme. Cependant toutes ensembles, elles définissant les éléments invariants de l'algorithme. La transformation, qui du cahier des charges a permis de construire le dossier d'analyse puis le programme n'est pas unique. La comparaison entre elles des représentations de l'algorithme abstrait, conduit à la connaissance des invariants communs à toutes les transformations analyse-programmation.

Il apparaît ainsi, que la comparaison directe d'une programmation particulière à un dossier d'analyse, ou même à un cahier des charges ne peut permettre une homologation efficace. Les invariants, déduits du programme et de ses autres représentations, formeront le premier terme de la comparaison.

Recherchons maintenant le second terme. Nous nous intéressons aux algorithmes de gestion, pour lesquels la connaissance de l'organisation et de la sémantique des fichiers d'entrées et de sorties du programme suffit à l'homme pour connaître les caractéristiques fonctionnelles de l'algorithme. Il s'agit là de la démarche initiale habituelle qui permet de formuler un problème de gestion :

> "Voici les fichiers dont on dispose et voici, par ailleurs, les résultats qu'il faut obtenir."

Ce que nous appelons les caractéristiques fonctionnelles sont alors fixées. Le cahier des charges, et plus encore le dossier d'analyse en sont des représentations impropres, parce que déjà orientées vers le programme ; elles risquent d'être erronées. Nous supposerons donc l'existence d'un dossier d'homologation qui contiendra seulement des informations sur l'organisation et la sémantique des fichiers d'entrées et de sorties du programme. Ce dossier formera le second terme de la comparaison qui réalise l'homologation.

Ce chapitre sera consacré à la recherche d'éléments invariants dans le programme. Nous utiliserons, pour cela, la représentation par tables de décisions à entrées limitées, et montrerons les aspects les plus originaux qu'autorise ce moyen de représentation. Les conclusions que nous poserons, resteront très incomplètes. Le problème de l'homologation d'un programme est un sujet suffisamment vaste pour justifier une étude séparée. Pratiquement nous pourrons réaliser un contrôle de la qualité de la programmation.

6.2 - MODULES DE TEST

La première étape du contrôle consistera à utiliser l'écriture généralement modulaire des programmes importants pour créer des modules de test. Chaque module de test est contrôlé séparément. Il est formé d'un ou plusieurs des modules fonctionnels. La séparation des modules de test est réalisée par l'utilisateur. L'accent peut être mis sur telle fonction principale qui sera alors isolée avec les routines qui la servent.

Une telle vérification indépendante de modules distincts est généralement entreprise lors du débourrage du programme. La difficulté tient alors à la création aussi simple que possible de modules qui créent l'environnement informatique du module à traiter. Dans notre cas, l'environnement est créé par le programme lui-même. D'où la possibilité de séparer, pour l'étude, les différentes parties d'un programme.

En pratique, la définition des modules qui serviront à l'homologation est entreprise sur la représentation fournie par l'ensemble des tables de décisions à entrées limitées. L'utilisation des directives, décrites au chapitre suivant, qui regroupent plusieurs tables, permet de créer une seule table de décisions, pour représenter le module de test. L'existence de sous-tables titrées autorise la présence de boucles dans le graphe de la table. C'est cette table unique du module de test qui servira de support à la recherche d'éléments invariants.

Nous étudierons successivement :

6.3 - Un jeu de valeurs d'essais,

6.4 - La hiérarchie des modules de traitements,

6.5 - La sémantique des entrées.

6.3 - JEUX D'ESSAIS, SEMANTIQUE DES TRAITEMENTS.

Il s'agit ici de contrôler les routines elles-mêmes du module. L'aide apportée par la table unique ou par le jeu de tables à entrées mixtes est importante, pour étudier la validité d'un module. Il s'agit là d'un des avantages, classiquement utilisé, de la représentation d'un programme à l'aide de tables de décisions. Cette vérification réalisée par l'homme peut encore se compléter par la construction, l'exécution et l'exploitation des résultats fournis par un jeu d'essais. Nous rendons compte dans les dernières pages de l'annexe I, de l'étude menée par la SESA, qui propose un algorithme d'affectation de valeurs aux variables, appelées d'état, d'une table de décisions classique. Il est ainsi

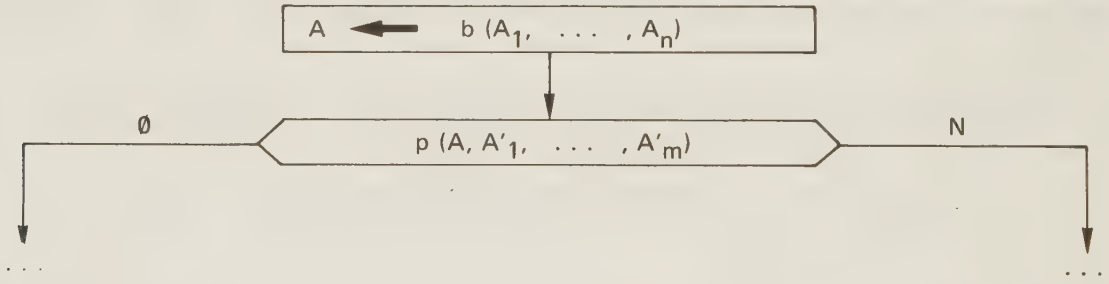
créé un essai pour chaque règle d'une table. Cet algorithme ne convient pas aux table de décisions uniques que nous avons définies. Rappelons que les différences tiennent :

- à la présence de ruptures sémantiques dans la table,
- à la présence de sous-tables titrées et de boucles dans la table.

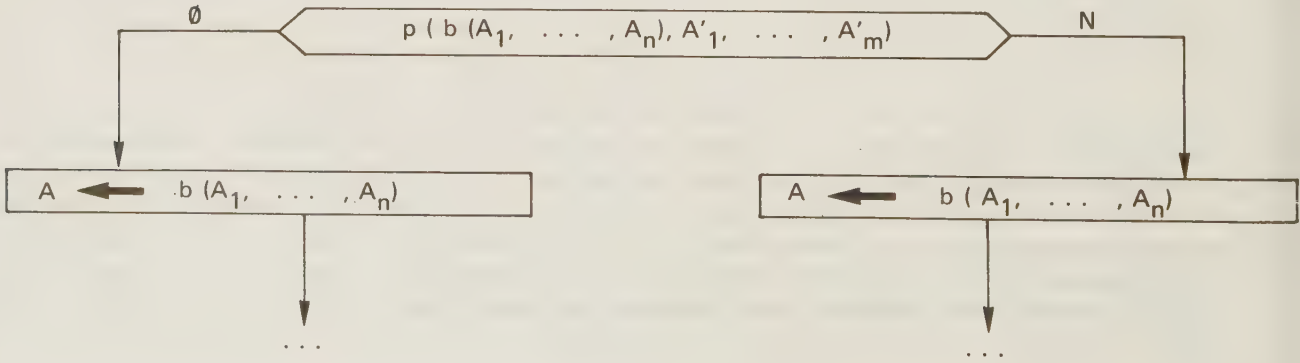
Ces différences modifiant l'exploitation d'une table de décisions et imposent d'interpréter au moins une partie de la souche et des entrées actions d'une sous-table avant d'évaluer les conditions de la sous-table suivante. Cette liaison interdit les degrés de liberté dans l'affectation de valeurs aux variables d'état, nécessaires au fonctionnement de l'algorithme. Nous chercherons à supprimer successivement les sous-tables d'édition, titrées ou non, de la table.

6.3.1 - Suppression des ruptures sémantiques

L'étude de la sémantique des conditions de la table unique permettra de supprimer le phénomène de rupture sémantique. Rappelons qu'une rupture sémantique se présente à l'occasion de l'évaluation d'une condition dont un opérande est une variable, réceptrice dans une assignation antérieure ; suivant le schéma :



Nous avons annoncé au chapitre 3 la transformation par substitution qui permet de supprimer la rupture de la table :



Pour illustrer cette transformation nous étudierons l'exemple du module suivant, emprunté à une gestion de stock. On dispose d'un élément d'entrée : OBJET formé d'une lettre et de 12 chiffres ; il caractérise un produit du stock.

- La lettre et les deux premiers chiffres indiquent la référence de l'article.

- Les quatre chiffres suivants représentent le nombre d'éléments en stock,

- Enfin les 6 derniers chiffres précisent la date (jjmmaa) de la dernière sortie d'un élément de l'article. Chacune de ces variables est soumise à des contrôles de validité, qui forment le module de test. Elles subissent ensuite un traitement qui n'est pas représenté ici.

Pour discuter le module envisagé, nous fournirons la table de décisions à entrées étendues qui forme une documentation de l'algorithme et la table unique qui permet de discuter l'homologation.

VERIFICATION-DES-DONNEES

LA LETTRE DU CODE EST	A	A	A	A	B	B	B	B	C	C	C	C	Autre
LE TYPE DU CODE EST	<20	<20	<20	⩾20	<30	<30	<30	⩾30	<40	<40	<40	⩾40	
LA DATE EST AU PLUS EGALE A LA DATE EN COURS	Ø	Ø	N		Ø	Ø	N		Ø	Ø	N		
ELLE EST > 10 03 71	Ø	N			Ø	N			Ø	N			
SIGNALER L'ERREUR DANS CHAQUE CAS		*	*	*		*	*	*		*	*	*	*
RETOUR DU MODULE		*	*	*		*	*	*		*	*	*	*
TRAITEMENT	*				*				*				

TABLE UNIQUE DU MODULE VERIFICATION-DES-DONNEES

LETTRE = 'A'	Ø	Ø	Ø	Ø	N	N	N	N	N	N	N	N	N
LETTRE = 'B'					Ø	Ø	Ø	Ø	N	N	N	N	N
LETTRE = 'C'									Ø	Ø	Ø	Ø	N

SOUS-TABLE 1

TYPE LESS 10 * (K + 1)	Ø	Ø	Ø	N	Ø	Ø	Ø	N	Ø	Ø	Ø	N	
DATE > DATE-DU-JOUR	Ø	N	N		Ø	N	N		Ø	N	N		
DATE > 710310		Ø	N			Ø	N			Ø	N		
MOVE 1 TO K	*	*	*	*									
MOVE 2 TO K					*	*	*	*					
MOVE 3 TO K									*	*	*	*	
DISPLAY OBJET													*
DISPLAY 'ERREUR DE LETTRE'													*

SOUS-TABLE 1

DISPLAY OBJET	*		*	*	*		*	*	*		*	*	
DISPLAY 'ERREUR DE DATE'	*		*		*		*		*		*		
DISPLAY 'ERREUR DE TYPE'				*				*				*	

RUPTURES DE SEQUENCE

EXIT FIN-VERIFICATION	*		*	*	*		*	*	*		*	*	*
GO TO TRAITEMENT		*				*				*			

La rupture sémantique dans la table unique est liée à la valeur de K. Dans chaque règle les assignations :

MOVE 1 TO K

MOVE 2 TO K

MOVE 3 TO K

sont réalisées avant l'évaluation de la condition :

TYPE LESS 10 * (K + 1)

La substitution indiquée conduit dans chacun des trois cas aux conditions :

TYPE LESS 20

TYPE LESS 30

TYPE LESS 40

et à la nouvelle table unique qui se présente sans rupture sémantique :

LETTRE = 'A'	Ø	Ø	Ø	Ø	N	N	N	N	N	N	N	N	N
LETTRE = 'B'					Ø	Ø	Ø	Ø	N	N	N	N	N
LETTRE = 'C'									Ø	Ø	Ø	Ø	N
TYPE LESS 20	Ø	Ø	Ø	N									
TYPE LESS 30					Ø	Ø	Ø	N					
TYPE LESS 40									Ø	Ø	Ø	N	
DATE > DATE-DU-JOUR	Ø	N	N		Ø	N	N		Ø	N	N		
DATE < 710310		Ø	N			Ø	N			Ø	N		
MOVE 1 TO K	*	*	*	*									
MOVE 2 TO K					*	*	*	*					
MOVE 3 TO K									*	*	*	*	
DISPLAY OBJET	*		*	*	*		*	*	*		*	*	*
DISPLAY 'ERREUR DE LETTRE'													*
DISPLAY 'ERREUR DE DATE'	*		*		*		*		*		*		
DISPLAY 'ERREUR DE TYPE'				*				*				*	

RUPTURES DE SEQUENCE

EXIT FIN-VERIFICATION	*		*	*	*		*	*	*		*	*	*
GO TO TRAITEMENT		*				*				*			

6.3.2 - Boucles de traitements

Après avoir réduit la difficulté liée aux ruptures sémantiques, il nous faut encore supprimer la présence des boucles internes au module de test ; les boucles sont représentées sur la table unique par la présence de sous-tables titrées. Une table de décisions classique autorise des boucles à la condition qu'elles référencent la table entière. Nous introduirons une transformation du graphe de la table unique qui modifiera la structure de chacune des boucles internes.

Un circuit est défini par un ensemble de sommets :

$$(X_1, X_2 \ldots X_n)$$

tels qu'il existe un arc (X_i, X_{i+1}) quelque soit i, et tels que $X_1 = X_n$.

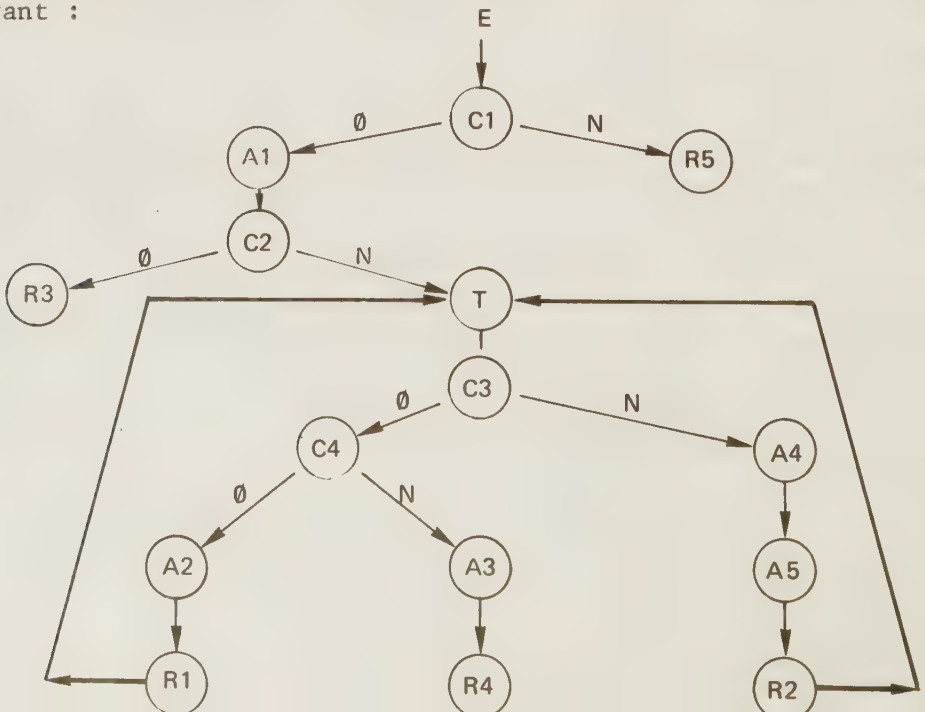
De même que dans le graphe d'un programme, nous définirons dans le cas du graphe d'une table de décisions :

- L'entrée E de la table est le titre du graphe de la table.

- Un accès au circuit est un chemin E-T tel que T appartient au circuit, aucun autre sommet de E-T n'appartient au circuit. La connexité du graphe de la table impose qu'au moins un tel accès existe.

- Rappelons que par définition T est le titre de la sous-table titrée, il est étiqueté par le nom de la sous-table. Il est de degré intérieur supérieur à 1.

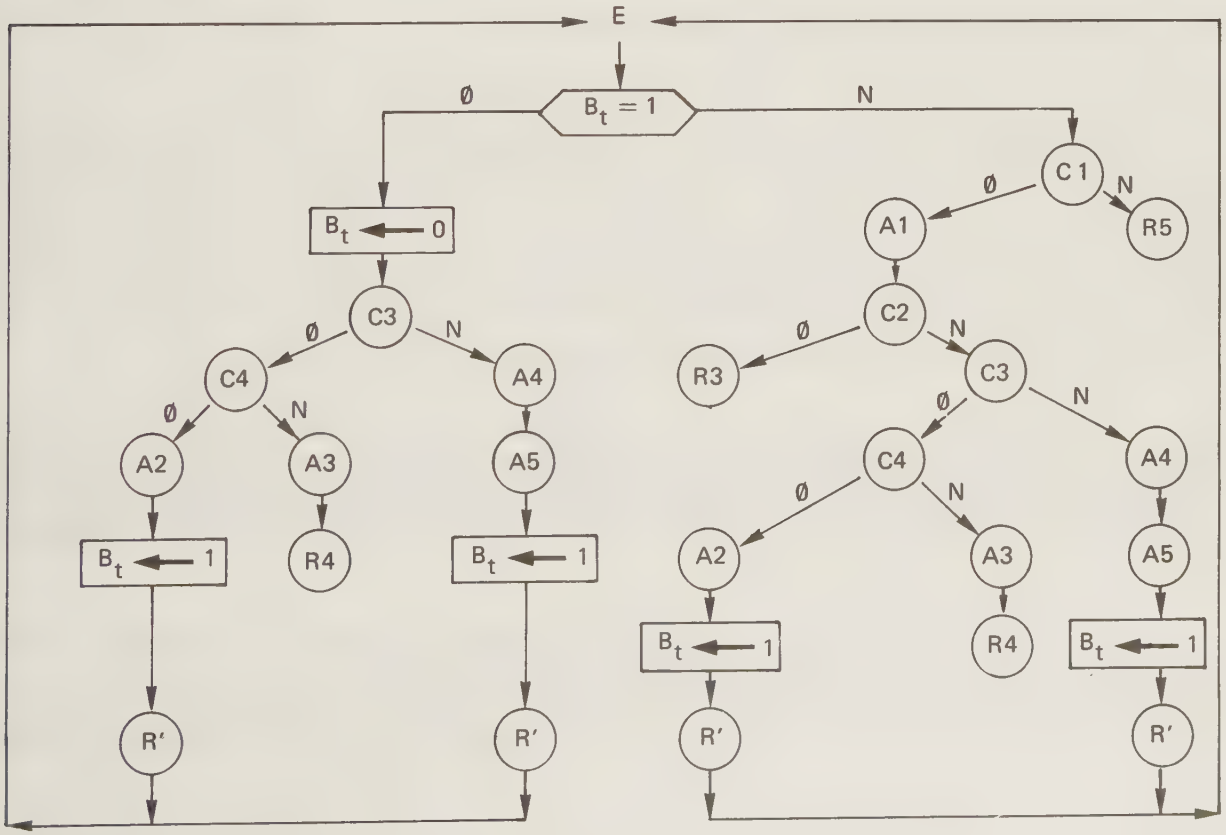
Nous nous intéresserons pour transformer le graphe aux arcs d'extrémités finales T, qui ne sont pas des accès ; ils sont dessinés en trait gras dans le schéma suivant :



Pour chaque sous-table titrée T de la table entière on crée une variable booléenne interne B_t, dont la valeur sera gérée comme suit :

- Une rupture de séquence qui référence la sous-table T est supprimée et remplacée par :
  - o une assignation : $B_t \leftarrow 1$
  - o une rupture de séquence vers le titre E de la table entière.
- Le titre de la table est modifié et remplacé par une batterie de conditions en séquence qui testent la valeur de chacune des variables B_t créées pour chaque sous-table titrée,
  - o La branche non (B_t vaut alors zéro) est reliée au test suivant (ou premier sommet de la table dans le cas de la dernière condition).
  - o La branche oui est suivie d'une assignation de remise à zéro : $B_t \leftarrow 0$.
  - o Assignation après laquelle on recopie la sous-table de titre T dont des ruptures de séquence ont été modifiées auparavant.
- Les sommets étiquetés par les noms des sous-tables T sont enfin supprimés.

Ainsi notre exemple devient :



Le graphe obtenu est sémantiquement équivalent au graphe de départ. C'est maintenant un graphe de table de décisions à entrées limitées, tel que défini au chapitre 3. Les circuits référencent la table entière.

Les deux transformations successives que nous venons de décrire :

suppression des ruptures sémantiques

suppression des boucles internes

ont permis de représenter le module entier à étudier, par une seule table de décisions qui organise un traitement arborescent. Toutes les sous-tables distinguées à l'édition ont été supprimées. La table est interprétée de façon classique :

évaluation des conditions

détermination de la règle

interprétation des actions

et peut alors faire l'objet de l'algorithme GAFT proposé par la SESA.

Le résultat sera un ensemble de valeurs d'essais pour chacune des règles de la table. Le contrôle du module sera en dernier ressort le fait de l'utilisateur qui comparera les fichiers résultats au dossier d'homologation que nous avons défini.

6.4 - HIERARCHIE DES MODULES DE TRAITEMENTS

L'évaluation d'un programme, ou d'un module de test, est très généralement limitée à la construction et l'exécution d'un jeu d'essais. Nous construirons maintenant, d'autres éléments qui permettront de contrôler la qualité de la programmation proposée, en recherchant des éléments invariants de la transformation analyse-programmation qui a fourni le programme. Nous montrerons que la création d'un jeu d'essais n'est pas le seul intérêt de la table de décisions unique qui représente pour nous le module à tester.

Le contrôle de la qualité d'un programme, ne peut être réalisé qu'en étudiant à la fois le programme et les valeurs sur lesquelles il agit. M. WARNIER dans ses ouvrages destinés à la formation des informaticiens, et qui décrivent la méthode L.C.P. étudie la hiérarchie des modules fonctionnels d'un programme de gestion.

On constate ainsi que la hiérarchie rencontrée entre les fichiers d'entrée d'un même programme, doit être retrouvée dans les modules qui traitent ces fichiers Ainsi à deux fichiers "imbriqués", par exemple un fichier séquentiel à mettre à jour et un fichier de mise à jour dont les enregistrements qui indiquent les mo-

difications, sont dans le même ordre, correspondront deux boucles de traitements imbriquées. La boucle de lecture du fichier de mise à jour devra être intérieure à la boucle de lecture du fichier principal. D'autres structures sont possibles pour réaliser cet exemple on peut affirmer qu'elles seront plus compliquées.

De même l'organisation des enregistrements d'un même fichier : enregistrements en séquence, chaînés, groupés par classe, etc., doit se retrouver dans l'organisation des modules qui traitent les enregistrements. Enfin, de la même façon, la structure des données élémentaires d'un enregistrement impose la structure des modules qui traitent chaque donnée, A des données répétitives correspondra une boucle dans le traitement ; à des données de nature différente un traitement arborescent..,

Nous attacherons le caractère d'invariant aux structures ainsi déduites directement de l'organisation des fichiers d'entrées. La recherche des hiérarchies de traitement indépendants de la programmation ne peut être envisagée avec la seule donnée du programme COBOL. Celui-ci ne précise à aucun moment l'organisation des fichiers entre eux ou des enregistrements d'un fichier. C'est la raison pour laquelle le dossier d'homologation doit indiquer très précisément la structure de chacun des fichiers du programme.

La recherche des éléments invariants de la structure des modules du programme peut être accomplie sur les organigrammes, souvent complexes. Nous montrerons qu'elle peut l'être de façon plus simple sur la table de décisions unique du module.

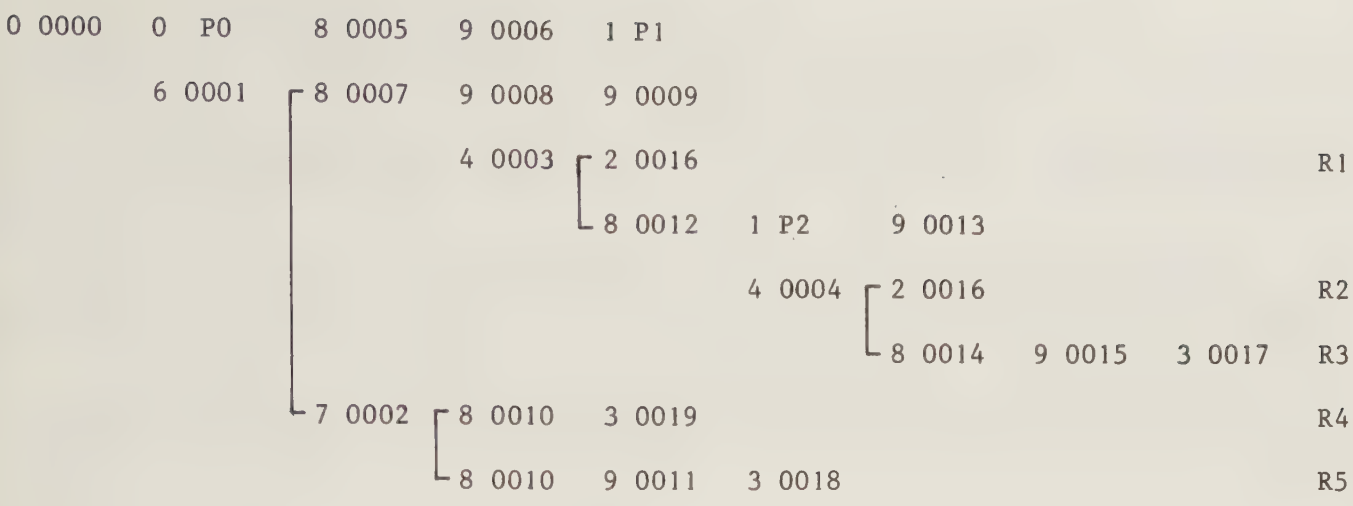
Nous illustrerons cette recherche par l'exemple du module suivant qui utilise au moyen d'un verbe PERFORM le module VERIFICATION-DES-DONNEES du paragraphe précédent. La lecture du programme est suffisamment aisée pour qu'il ne soit pas besoin de le décrire davantage. Nous avons supposé que la variable de groupe D est décomposée :

- en un FILLER auquel est attaché le nom-condition BLANC
- un nombre D1 qui pointe le dernier élément écrit du tableau D2
- un tableau D2, dont chaque élément représente un article de stock de l'exemple précédent.

Ce programme peut être représenté par la seule table de la page suivante. Pour information le codage correspondant à cette table est le suivant :
Les références sont celles des lignes de la table.

Table unique du module PO

	SOUS-TABLE P1					
1	ONE MORE RECORD IN LECTURE	Ø	Ø	Ø	N	N
2	I GREATER 1000				Ø	N
	SOUS-TABLE 2					
3	FILLER = SPACE	Ø	N	N		
	SOUS-TABLE P2					
	SOUS-TABLE 4					
4	N GREATER D1		Ø	N		
5	OPEN INPUT LECTURE OUTPUT ECRITURE	*	*	*	*	*
6	MOVE ZERO TO I	*	*	*	*	*
	SOUS-TABLE P1					
7	READ LECTURE RECORD	*	*	*		
8	ADD 1 TO I	*	*	*		
9	MOVE CARTE TO I	*	*	*		
10	CLOSE ECRITURE LECTURE				*	*
11	STOP 'CHANGER CARTES'					*
	SOUS-TABLE 2					
12	MOVE ZERO TO N		*	*		
	SOUS-TABLE P2					
13	ADD 1 TO N		*	*		
	SOUS-TABLE 4					
14	MOVE D2 (N) TO OBJET			*		
15	PERFORM VERIFICATION-DES-DONNEES THRU FIN-VERIFICATION			*		
	RUPTURES DE SEQUENCE					
16	GO TO P1	*	*			
17	GO TO P2			*		
18	GO TO PO					*
19	STOP RUN				*	



0 0000

Et le programme correspondant :

```
PO.
        OPEN INPUT LECTURE OUTPUT ECRITURE.
        MOVE ZERO TO I.
P1.
        READ LECTURE AT END GO TO FIN,
        ADD 1 TO I. MOVE CARTE TO D.
        IF BLANC GO TO P1.
        MOVE ZERO TO N.
P2,
        ADD 1 TO N. IF N GREATER D1 GO TO P1.
        MOVE D2 (N) TO OBJET, PERFORM VERIFICATION-DES-DONNEES
        THRU FIN-VERIFICATION. GO TO P2.
FIN.
        CLOSE ECRITURE LECTURE.
        IF I GREATER 1000 STOP RUN ELSE
        STOP 'CHANGER CARTES' GO TO PO.
```

Pour étudier la structure du module, nous nous intéresserons aux ruptures de séquence de la table. La présence de deux sous-tables titrées : P1 et P2 associées aux ruptures de séquence des lignes 16 et 17, permet de conclure à la présence de boucles internes dans l'organigramme. Nous pouvons encore affirmer une boucle P0 du module entier : ligne 18. Et enfin nous pouvons affirmer qu'il n'y en a pas d'autres.

La donnée de la table unique permet de donner la liste complète des boucles du module ; quelle que soit la taille du module la recherche des boucles reste simple ; elle consiste à comparer les titres de la table et des sous-tables aux références des ruptures de séquence qui sont toutes regroupées en fin de table. Une représentation par organigramme ne permet pas cette recherche simple des boucles. Elle ne permet surtout pas d'affirmer que la liste construite est exhaustive.

Nous devons, maintenant rechercher la nature des boucles P0, P1 et P2, aux fins de les comparer à l'organisation des fichiers d'entrées indiquée dans le dossier d'homologation.

La première action de la boucle P0 (ligne 5) est une ouverture de fichier. A cette boucle doit correspondre une organisation répétitive du fichier d'entrée. Nous supposerons que le dossier d'homologation stipule que le programme doit être exécuté pour plusieurs fichiers dont seul le dernier a plus de 1 000 cartes. Nous conclurons donc au caractère invariant de la boucle P0.

De même, la première action (ligne 7) de la sous-table P1 conduit à une structure répétitive des enregistrements du fichier d'entrée ; elle est bien vérifiée. D'où le caractère invariant de la boucle P1. Remarquons encore que la sous-table P1 est "imbriquée" (au sens défini au chapitre 3) dans la table entière P0. C'est encore un élément invariant, lié à la structure d'imbrication des enregistrements du fichier de lecture.

Les mêmes conclusions pourraient être posées à propos de la boucle P2, imbriquée dans la boucle P1. Cette structure recopie la structure des données élémentaires D2 :

- répétitives, d'où la boucle P2,
- incluses dans un enregistrement, d'où l'imbrication des sous-tables P2 et P1.

Nous avons ainsi montré sur un exemple que la table de décisions unique, loin de masquer la structure du programme, la rend plus accessible. La comparaison, faite par l'homme, de cette structure et de la structure des entrées du programme permet un contrôle supplémentaire de la qualité de la programmation.

6.5 - SEMANTIQUE DES ENTREES DU PROGRAMME

Après avoir comparé d'une part, le contenu des modules de traitements du programme et la sémantique des fichiers de sortie par l'intermédiaire de jeux d'essais, d'autre part la structure hiérarchique des modules et celles des fichiers d'entrée du programme, il nous reste à étudier la sémantique des conditions rencontrées dans le module de test pour la comparer à la sémantique des entrées de ce module, telle qu'elle est décrite dans le dossier d'homologation.

L'homologation est ici encore, le fait de l'homme, qui analyse dans une même étude le programme et les valeurs sur lesquels il agit. L'ordinateur interviendra comme un accessoire efficace.

Le module de test est représenté par la table unique construite lors de l'élaboration du jeu d'essais, c'est-à-dire après suppression des ruptures sémantiques et des boucles internes. Nous chercherons à créer une partition des valeurs d'entrées. A chaque pavé ou domaine de cette partition correspond une règle de la table unique et un traitement original propre.

L'étude formelle et automatique des conditions exprimées en fonction des seules variables d'entrées génère les bornes de chaque domaine. Exprimés de façon indépendante des intermédiaires de programmation, ces domaines apparaissent comme un élément invariant de l'algorithme.

L'intervention de variables extérieures à la programmation (les bascules B_i) impose que la partition construite ne traduit pas complétement la complexité de l'algorithme. Une telle partition permet cependant de diriger le travail de recherche systématique des erreurs. En l'absence d'aucune autre information déterministe, il ne peut être exécuté que par l'homme.

- Omission de tests de vérifications de validité des données. Ces omissions sont mises en lumière par l'étude des domaines dont les limites sont des bornes de l'ensemble total, bornes qui auront pu être données systématiquement trop larges.

- Illogismes dans la sémantique des tests qui conduisent à des domaines vides, correspondants à des contradictions sémantiques. Cette vérification peut encore être réalisée par l'introduction dans la table des entrées implicites ⁑ et § proposées par S. L. Pollack (cf. Chap. 2). Etre conduit à imposer une entrée ⁑ (équivalente à N) sur une entrée explicite Ø de la table signale un domaine inaccessible et vide.

- Erreurs sur l'appartenance des bornes d'un domaine à celui-ci. Ces erreurs fréquentes et généralement peu décelables sont ainsi mises en valeur par l'édition claire des bornes des pavés.

- Erreurs dans la sémantique des tests qui conduit à distinguer des pavés différents, alors que le traitement des valeurs d'entrées prises dans ces pavés aurait dû être identique.

- Erreurs inverses. Un même domaine recouvre deux ou plusieurs cas pour lesquels le traitement doit être différent. Il s'agira en général, dans ces deux derniers cas, d'erreurs d'analyse de l'algorithme. Un jeu d'essais choisis par l'homme, et a fortiori, générés automatiquement d'après le programme lui-même, ne décèlent pas de telles erreurs, dont la correction n'est pas inscrite dans la représentation utilisée de l'algorithme abstrait désiré.

Cette liste des erreurs décelables, après une étude dirigée par l'homme, sur la table de décisions unique, ne prétend pas être complète. Toutes les informations contenues dans une table de décisions n'y ont pas été exploitées.

6.6 - CONCLUSION

Les derniers paragraphes de ce chapitre ont montré que la construction automatique d'un jeu d'essais ne peut fournir à elle seule une homologation suffisante. L'homme interviendra dans le choix des essais qui seront exécutés effectivement. L'homme interviendra encore dans l'exploitation des résultats. L'homologation sera le produit direct de cette intervention.

En l'absence de tout autre moyen efficace d'homologation, le jeu d'essais reste la méthode universellement employée. L'ambition de ce chapitre est d'apporter quelques critères supplémentaires de contrôle qualitatif d'une programmation particulière.

Nous nous sommes attachés à montrer que l'étude de la validité d'un programme ne peut séparer l'ensemble formé par le programme et ses valeurs de travail. Ainsi le dossier d'homologation que nous avons défini, ne contient d'information que sur les fichiers utilisés dans le programme, à l'exclusion de tout autre renseignement sur le programme lui-même. Ces renseignements sont susceptibles d'être entachés d'erreurs.

Enfin nous avons souligné l'intérêt de la représentation par table de décisions dans l'homologation du module, alors même que la taille de la table utilisée détruit tout avantage documentaire.

CHAPITRE 7

UTILISATION DE LA CHAINE TABOL

Les programmes de la chaîne TABOL ont été mentionnés à différentes reprises dans les chapitres précédents. Il faut distinguer :

- Les programmes qui construisent de façon automatique les tables de décisions à entrées limitées traduisant un programme donné.

  - PD1A analyse les trois premières divisions.
  - PD1B analyse la PROCEDURE DIVISION.
  - PD1C construit les tables de paragraphes et recherche les entrées de circuits.
  - PD2A regroupe les tables de paragraphes.
  - PD2B recherche les ruptures sémantiques et tranforme les phrases de codage en phrases syntaxiquement correctes pour la représentation des tables à entrées étendues.

- Les programmes qui réalisent l'édition des tables :

  - PD3A construit l'ordre d'édition des actions et des conditions.
  - PD3B édite les tables sur l'imprimante.

- Enfin les programmes de traitement des directives d'édition. Ils construisent les tables de décisions à entrées étendues.

  - PD3C lit toutes les directives et traite les directives que nous qualifierons de sémantiques (regroupement d'actions et de conditions).
  - PD3D traite les directives de structure définies dans la suite du chapitre.

7.1 - CREATION DES FICHIERS

Deux programmes successifs n'échangent d'informations qu'à travers des fichiers catalogués sur mémoire de masse. La plupart d'entre eux sont temporaires, quelques uns sont permanents. Ceux-ci permettent de conserver en mémoire une image codée des tables éditées, de façon à reprendre, sous contrôle de directives données ultérieurement, l'édition d'un nouveau cycle de tables de décisions.

Le tableau ci-dessous indique dans les colonnes successives :

1) Le nom externe du fichier,

2) Le nombre de caractères de l'enregistrement logique.

3) Le facteur de blocage dans la version UNIVAC 1108 de TABOL.

4) Le caractère temporaire (T) ou permanent (P) du fichier.

5) Le nombre d'enregistrement logiques souhaitable ou obligatoire (0).

6) La colonne 6 indique les fichiers qui, pour des raisons d'homogénéité doivent être de taille identique.

7) Le mode d'accès : séquentiel (S) ou direct (D).

8) Enfin la dernière colonne suggère le contenu du fichier, pour permettre d'évaluer pour chaque programme COBOL à traiter la taille nécessaire du fichier.

1	2	3	4	5	6	7	8
HASH	34	3	T	401 0	⌐	D	Table des hash-code des noms
DICT	8	11	T	401 0	⌙	S	de variables.
CTES	64	1	T	100		S	Noms conditions et constantes.
MASK	7	15	T	8000	⌐	S	Codage des tables.
DEF	7	15	P	8000	⌙	S	"
DO	7	15	P	500		S	Tables des PERFORM ... UNTIL
PARAG	34	3	P	203 0		D	Titres des tables.
BOR	31	3	T	500		S	Liaisons entre tables paragraphe
TAB	110	1	P	500	⌐	D	Souches actions et conditions.
INREF	110	1	T	500	⌙	D	"
REFE	110	1	T	500	⌙	D	"
ENT	95	1	P	500	⌐	D	Entrées.
ANT	95	1	T	500	⌙	D	"
FIDOM	33	3	T			D	Directives de structure.

7.2 - APPEL DES PROGRAMMES D'ANALYSE SYNTAXIQUE

Pour illustrer l'emploi de la chaîne TABOL, nous présenterons ici l'exemple dont les résultats d'exécution forment l'annexe II. Le programme a été choisi relativement court, pour éviter que le volume des documents fournis, soit excessif. Il s'agit d'un programme de contrôle des cartes de mise à jour hebdomadaire d'un fichier. L'entreprise a un certain nombre d'employés. Chacun d'eux travaille alternativement au profit de différents contrats, dont l'entreprise a la charge. Les cartes qui font l'objet du contrôle indiquent pour chaque employé et par demi-journée, les contrats dont il s'est occupé.

On trouvera dans l'annexe :

A II - a. Le listing des trois premières divisions ; exécution de PD1A.

A II - b. Le listing de la PROCEDURE DIVISION.

A II - c. Les 19 tables à entrées limitées qui ont été éditées.

A II - d. Un fichier de directives de transformation. Pour des raisons de volume, celui-ci a été limité aux six premières tables.

A II - e. Les six tables à entrées mixtes qui en résultent.

Un seul cycle de table à entrées étendues est fourni. La documentation annexée est donc très incomplète.

Dans la version présentée ici, le programme à traiter est lu sur cartes. L'appel des deux premiers programmes PD1A et PD1B d'analyse syntaxique se fait comme suit :

- Carte d'exécution du programme PD1A.

```
IDENTIFICATION DIVISION.
...
ENVIRONMENT DIVISION.
...
DATA DIVISION.
...
```

- Carte fin de fichier.
- Carte d'exécution du programme PD1B.

PROCEDURE DIVISION.

...

- Carte fin de fichier.

La carte de fin de fichier qui termine les trois premières divisions peut être omise si une carte PROCEDURE DIVISION est présente ; toutes les cartes suivantes seront alors traitées comme des commentaires par le programme PD1A. La carte d'exécution du programme PD1B peut être suivie du programme entier. PD1B recherchera dans ce cas le début de la procedure division. Cette disposition a été adopté pour qu'il soit possible de dupliquer le programme sur cartes et de placer chacun des deux exemplaires derrière les cartes d'exécution de PD1A et PD1B.

Il est important que le programme donné soit syntaxiquement correct. Les programmes PD1A et PD1B ne sont pas des reconnaisseurs syntaxiques COBOL complets et une erreur de syntaxe entrainera des résultats imprévisibles. Un avantage de ce caractère est que tous les COBOL sont acceptés puisque non complètement analysés.

7.3 - APPEL DES PHASES PD1C ET PD2A.

La phase PD1C lit une première carte qui autorise trois options de contrôle du regroupement des paragraphes. L'option choisie est lue dans la première colonne de la carte :

- Option R.

Elle signifie que les entrées de circuits et noeuds de losanges sont recherchés par programme. Rappelons qu'à chaque entrée de circuit et chaque paragraphe début de module appelé par PERFORM correspondra un titre de table de décisions. Derrière cette carte d'option R, on peut indiquer des noms de paragraphes auxquels devront correspondre un titre de table de décisions. Un nom de paragraphe par carte est indiqué sur les 30 premières colonnes de la carte, cadré à gauche sans blanc. La dernière carte doit être une carte de fin de fichier. Elle est suivie de la carte d'appel de l'exécution de PD2A.

L'exemple fourni en annexe a été exécuté avec l'option R. Les paragraphes : INIT (table 3), 1B25 (table 8), 2A09 (table 12), 2A10 (table 13), 2A20 (table 16) ont été signalés comme étant des têtes de modules de traitement.

- Option C.

Les entrées de circuits ne sont pas recherchées par le programme. Les paragraphes dont les noms suivent (sous le même format que dans l'option R) sont les entrées de circuits désirées. Il est important de désigner au moins un paragraphe sur chaque circuit du programme. Les paragraphes têtes de modules appelés par un verbe PERFORM n'ont pas à être désignés il correspondront à une table éditée. Comme dans le cas de l'option R la carte fin de fichier est suivie de la carte d'appel de PD2A.

- Autre option.

Si la première colonne de la première carte contient un autre caractère que R ou C, à chaque paragraphe du programme correspondra une table de décisions. Il est inutile de faire suivre cette option de noms de paragraphes. La phase PD2A ne doit pas être exécutée. Cette phase est en effet celle qui regroupe les paragraphes dans une même table.

7.4 - APPEL DE LA PHASE PD2B.

Comme le programme PD2A, le programme PD2B n'a aucune carte donnée. Il recherche les ruptures sémantiques dans les tables précédemment construites.

7.5 - EDITION DES TABLES

L'édition des tables est provoquée par les exécutions successives de PD3A et PD3B. Ces programmes n'ont besoin d'aucune carte donnée. Ils chargent les fichiers permanents de façon à permettre l'interruption du traitement et sa reprise après construction des directives d'édition des tables de décisions à entrées étendues.

7.6 - TABLES DE DECISIONS A ENTREES MIXTES

Le fichier des directives d'édition des tables de décisions à entrées étendues est lu par le programme PD3C. Les directives dites sémantiques :

ACT, COND, SUPR sont traitées par ce même programme. Les autres directives dites directives de structure sont traitées par le programme PD3D.

Le fichier des directives est articulé en modules de directives. Chaque module est propre à une table et regroupe les directives qui doivent lui être appliquées. Chaque module est précédé d'une carte indiquant le numéro de la table au cycle d'édition précédent. Ce numéro est cadré à droite sur les 4 premières colonnes de la carte. Les tables doivent être traitées en ordre de numéro croissant. Si pour une table aucune directive n'est prévue, la carte indiquant son numéro peut être omise.

La directive SUPR, s'il doit y en avoir une, doit être la première directive du module. L'ordre dans lequel sont écrites les autres directives n'importe pas.

L'annexe II - d indique un exemple de fichier de directives tel qu'il a été édité au cours de l'exécution du programme PD3C.

7.6.1 - Syntaxe des directives

Chaque directive est caractérisée par un code opératoire alphanumérique suivi immédiatement (sans blanc) d'une virgule. Les codes disponibles sont indiqués ci-dessous :

- Directives sémantiques : elles sont au nombre de trois.
  - SUPR
  - COND
  - ACT
- Directives de structure : ce sont les cinq suivantes,
  - AJRS
  - REG
  - SEP
  - NUL
  - CHGT

Le texte qui suit le code opératoire est divisé en champs, séparés par des virgules. Chaque champ peut être divisé en zones séparées par des slash (/). Ces deux caractères de séparation : ',' et '/' peuvent être modifiés ; ils sont représentés dans le programme PD3C par les variables alphanumériques, SCH et SEZ. La directive se termine par un point.

Une directive peut occuper plusieurs cartes successives. Les quatre premières colonnes des cartes suites doivent être laissées blanches. Les blancs sont partout ignorés, à l'exception des deux cas suivants :

- Pas de blanc entre le code opératoire et la virgule qui le suit.

- Dans les textes de souches une succession de blancs est partout traitée comme un seul blanc. Dans le texte que construit le programme les mots ne seront séparés que par un seul blanc.

7.6.2 - Directive COND

COND, C1 [/C2/ ... Cn], texte de souche sans virgule, E1 [/E2 ... /Em], R_{11} [/R_{12} ... /R_{1i}], R_{21} [/R_{22} ... /R_{2i}], ... R_{m1} [/R_{m2} ... /R_{mi}].

Les zones entre [...] sont optionnelles.

Cette directive, d'un emploi délicat, a pour effet de regrouper sur une même ligne, des conditions d'une même sous-table d'édition. Ces conditions doivent être sémantiquement liées (voir définition chapitre 5).

Les zones C1, ... Cn précisent les numéros de lignes des conditions à regrouper. Si une seule ligne y est mentionnée, la directive aura pour effet de modifier texte et entrées de cette condition.

Le champ suivant contient le texte de la souche de la condition regroupée. Il ne doit pas contenir de séparateur de champ (virgule).

Les zones E1 ... Em précisent les entrées étendues de la condition regroupée. Elles ne peuvent avoir plus de trois caractères non blancs. Il doit y en avoir autant que de règles dans l'ensemble de règles complémentaires.

Les champs suivants fournissent les numéros de règles complémentaires. L'entrée précisée en E1 sera éditée sur les règles trouvées en R_{11}, R_{12} ... Il doit y avoir autant de champs règles différents qu'il y a d'entrées différentes. Un ensemble de règles complémentaires est donné en écrivant une règle par champ.

On trouvera dans l'exemple déjà cité en annexe, plusieurs utilisations de la directive COND. Pour préciser d'avantage l'emploi de cette directive, nous traiterons les exemples suivants.

Soit la table à entrées limitées :

1	C1	Ø	Ø	Ø	Ø	N

SOUS-TABLE 1

2	C2	Ø	Ø	N	N	
3	C3			Ø	N	
4	C4	Ø	Ø			
		R1	R2	R3	R4	R5

La directive COND, 2/3, C O, e1/ e2 /e3, 1,3,4. aura pour effet de permettre l'édition de la table suivante :

1	C1	Ø	Ø	Ø	Ø	N

SOUS-TABLE 1

2	CO	e1	e1	e2	e3	
3	C4	Ø	N			
		R1	R2	R3	R4	R5

L'entrée e1 est écrite sur les règles 1 et 2 sans qu'il ait été nécessaire de préciser ces deux règles.

Soit maintenant la table :

1	C1	Ø	Ø	Ø	Ø	N	N	N
2	C2	Ø	Ø	N	N	Ø	N	N
3	C3			Ø	N		Ø	N
4	C4	Ø	N					
		R1	R2	R3	R4	R5	R6	R7

On remarque que les entrées des lignes C2 et C3 se retrouvent toutes identiques deux fois. Il y a deux ensembles de règles complémentaires, ils devront être précisés tous les deux dans la directive pour permettre de regrouper C2 et C3. La directive sera :

COND, 2/3, CO, E1 / E2 / E3, 1/5, 3/6, 4/7. et la table éditée :

1	C1	Ø	Ø	Ø	Ø	N	N	N
2	CO	E1	E1	E2	E3	E1	E2	E3
3	C4	Ø	N					
		R1	R2	R3	R4	R5	R6	R7

En fin dans l'exemple de la table suivante, il faudra deux directives pour regrouper les lignes C2 et C3.

1	C1	Ø	Ø	Ø	Ø	N	N	N
2	C2	Ø	Ø	N	N	Ø	Ø	N
3	C3			Ø	N	Ø	N	
4	C4	Ø	N					
		R1	R2	R3	R4	R5	R6	R7

Les entrées des règles R1 à R4 ne se retrouvent pas identiques dans les règles R5 à R7. Il est apparu difficile d'imaginer que cela puisse correspondre à une seule condition et 6 entrées différentes dans un cas réel. Les deux directives possibles sont :

COND, 2/3, CO, E1 / E2 / E3, 1, 3, 4.

et

COND, 2/3, C'O, E'1 / E'2 / E'3, 5, 6, 7.

1	C1	Ø	Ø	Ø	Ø	N	N	N
2	CO	E1	E1	E2	E3			
3	C'O					E'1	E'2	E'3
4	C4	Ø	N					
		R1	R2	R3	R4	R5	R6	R7

7.6.3 - Directive ACT

ACT, A1 [/ A2 / ... / An] , texte de souche sans virgule, E1 [/ ... Em] , R11 [/ R12 ... / R1i] ,[R21 [/ ... / R2j] , ... , [Rm1 / ... / Rml]] .

Cette directive a pour objet de regrouper sur une seule ligne les actions d'une même sous-table dont les numéros sont indiqués en A1 ... An.

Le nouveau texte désiré est donné dans les champs suivants. Ce texte ne doit pas comporter de virgule. Le premier mot, tronqué éventuellement après le 9ème caractère, sera traité comme le verbe de l'instruction, sauf si le texte de souche commence par un /.

Les entrées font l'objet du champ suivant. L'entrée Ei se retrouvera sur les règles Ri1 ... Rii' précisées dans le ième champ suivant.

Le résultat de la directive ACT est très intuitif. Les règles indiquées doivent toutes avoir une entrée sur l'une au moins des actions à regrouper. On ne peut demander deux entrées différentes pour deux règles dont les entrées sur les actions sont identiques. Il est déconseillé d'appliquer la directive ACT à des instructions de ruptures de séquence.

La directive ACT permet deux transformations différentes quant à la logique du programme. Elles sont illustrées par les deux exemples suivants.

Exemple 1

		R1	R2	R3	R4
13	A1	*	*		*
14	A2	*	*		*
15	A3			*	*
16	A4	*	*		*
17	A5			*	*
18	A6			*	*
19	A7			*	*

Les directives :

ACT, 13 / 14 / 16, / Traitement 1, * , 1 / 2 / 4.

et :

ACT, 15 / 17 / 18 / 19 , REALISER LE TRAITEMENT 2, * , 3 / 4 .

auront pour effet de transformer la table en la suivante : (remarquer l'effet du / écrit avant traitement 1 dans la première directive).

		R1	R2	R3	R4
13	TRAITEMENT 1	*	*		*
14	REALISER LE TRAITEMENT 2			*	*

La directive est ici employée pour regrouper des actions en séquence sous un même libéllé résumé. Le deuxième emploi correspond à celui qui est illustré par 1'exemple suivant :

Exemple 2

		R1	R2	R3
21	MOVE 1 TO J	*		
22	MOVE 2 TO J		*	
23	MOVE 3 TO J			*

La directive : ACT, 21 / 22 / 23 , MOVE ... TO J, 1 / 2 / 3 , 1/2/3. entrainera 1'édition de la table suivante :

		R1	R2	R3
21	MOVE ... TO J	1	2	3

7,6,4 - Directive SUPR

SUPR, numéro [/ numéro ...] .

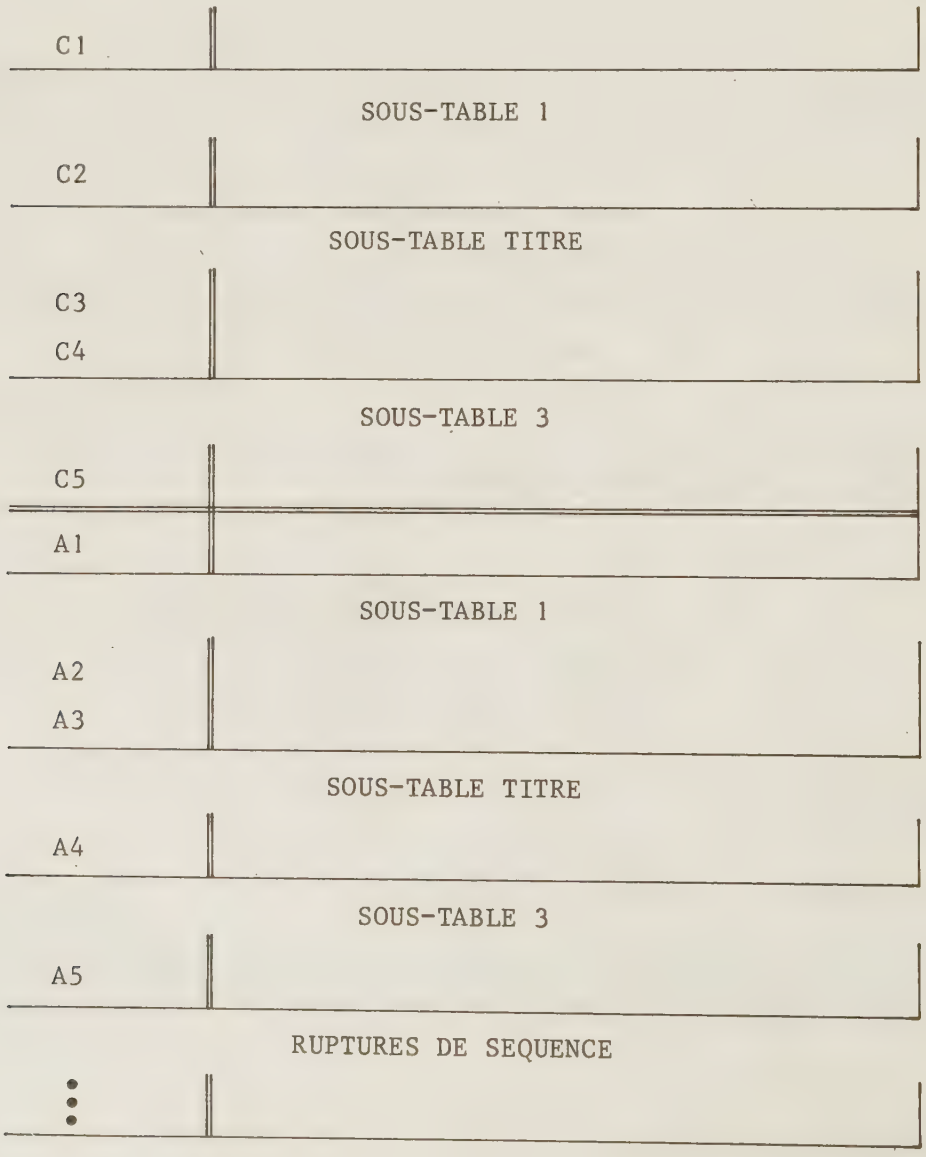
L'exécution de cette directive a pour effet de supprimer une rupture sémantique, et la singularisation d'une sous-table à 1'édition. La sous-table

est désignée par son numéro d'ordre. Rappelons que celui-ci est édité en clair si la sous-table n'est pas titrée.

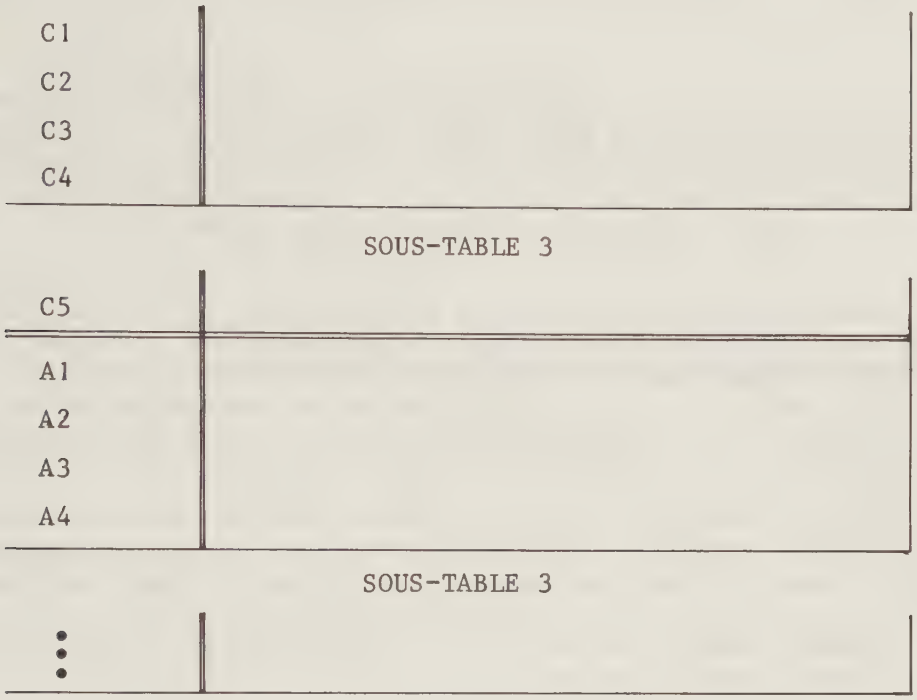
Aux sous-tables titrées, correspond également un numéro d'ordre. Une directive SUPR qui fait référence à un tel numéro non édité conduira à supprimer le titre de la sous-table. Aucune instruction de rupture de séquence ne devra plus faire référence à cette sous-table.

La directive SUPR doit être la première directive traitée pour chaque table. Une seule directive suffit. Le résultat de la directive SUPR est immédiat. Les directives COND et ACT qui la suivent peuvent mentionner des actions ou conditions appartenant aux deux sous-tables qui viennent d'être réunies, comme appartenant à la même sous-table.

Soit par exemple la table :



Une directive SUPR, 1 / 2. aura pour effet l'édition de la forme :



7.6.5 - Directives AJRS

Cette directive a l'effet inverse de la précédente. Elle permet d'ajouter la mention d'une rupture sémantique, c'est-à-dire de créer une sous-table d'édition.

AJRS, numéro de condition [, ...] .

Il y est indiqué le numéro de la condition qui devient le titre de la sous-table. Cette condition ne doit faire l'objet d'aucune directive COND dans le même fichier de directives. La directive ne peut comprendre plus de 30 caractères non blancs, non compris les quatre caractères du code et la virgule. Une seule directive est autorisée.

7.6.6 - Directive REG

Une telle directive impose que la table à laquelle elle s'applique soit regroupée à toutes les tables qui l'appellent. Deux options sont possibles :

1 - REG, T [, nouveau titre de la table] .

2 - REG, S .

Dans le premier cas : option T (pour titre) la table conserve un titre qui sera édité après les mots SOUS-TABLE et à la place du numéro d'ordre. C'est cette option qui permet la création de sous-tables titrées. Il est ensuite possible qu'un circuit s'ouvre sur cette table. On trouvera alors des instructions de ruptures de séquence référençant la sous-table titrée.

Le champ suivant l'option titre peut contenir un nouveau titre pour la table. Toutes les ruptures de séquence qui référençaient l'ancien titre seront alors modifiées automatiquement. Le nouveau titre ne doit pas avoir plus de 30 caractères non blancs. En cas de conflit entre une directive REG utilisée avec l'option T et une directive CHGT, c'est la directive REG qui détermine le titre de la sous-table.

On trouvera une directive complète REG, T , titre. appliquée à la deuxième table de l'exemple signalé en annexe. Le fichier MINI-AFF contient une partie des informations contenues dans le fichier ML-AFF des contrats de l'entreprise : le numéro de chaque contrat et la date de fin de contrat. La boucle de construction de ce fichier réduit comprend :

- Une initialisation : ouverture des fichiers.

- Une boucle : lecture séquentielle du fichier principal, mouvements des informations, écriture d'un enregistrement.

La présence d'une boucle a imposé la décomposition en deux tables à entrées limitées. Il a paru plus clair, ensuite, de décrire ce premier module par une seule table, au moyen de la directive :

REG, T , ENREGISTREMENT-RESUME.

appliquée à la table FAB-MINI, qui ainsi change de nom.

Dans le deuxième cas : option S (pour simple) la table est regroupée sur toutes celles qui l'appellent. Le regroupement s'opère sans préciser de titres aux sous-tables créées.

La table d'entrée du programme ne peut faire l'objet d'une directive.

Les instructions qui provoquent le regroupement peuvent être les GO TO et les verbes PERFORM qui n'ont à aucun niveau précédent fait l'objet d'une directive ACT. L'instruction d'appel doit donc être restée identique à ce qu'elle était dans l'édition des tables à entrées limitées, ou n'avoir été modifiée que par l'intermédiaire de la directive CHGT.

Si l'instruction d'appel au regroupement est un verbe PERFORM quelques précautions doivent être prises, Elles sont détaillées par les exemples suivants.

- 1er cas PERFORM P1

La table P1 est regroupée avec ou sans titre. Elle peut faire ou non l'objet d'une directive NUL.

- 2ème cas PERFORM P1 THRU P2

La partie du programme codée par les tables fermées P1 à P2 doit :

- ou être représentée par une seule table. Cette table fait l'objet de la directive REG.
- ou être représentée par plusieurs tables ; et dans ce cas il est nécessaire que toutes les tables soient l'objet d'une directive REG. Une directive REG appliquée à la table P1 seule, conduit à des résultats imprévisibles.

- 3ème cas PERFORM P1§0128 ou
PERFORM P1§0130§P2

Des instructions de cette forme sont générées pour chaque instruction COBOL du type : PERFORM ... UNTIL ...

Une table est créée qui résument les conditions de l'option UNTIL et les appels à la séquence de traitement P1 ou P1 THRU P2. Ces tables intermédiaires fermées peuvent faire l'objet d'une directive REG. L'option T est nécessaire.

7.6.7 - Directive SEP

Cette directive provoque l'édition en tables physiquement distinctes d'une table et d'une de ses sous-tables d'éditions. La sous-table titrée ou non dont on désire l'impression séparée, est indiquée par son numéro d'ordre à l'édition, compte tenu des éventuelles directives AJRS et SUPR appliquées à la table.

Si la sous-table n'est pas titrée, le champ suivant le numéro doit indiquer un titre de 30 caractères au plus, les blancs sont ignorés.

SEP, ordre [, titre].

Il faut remarquer que si la sous-table ainsi distinguée par son ordre n'est pas titrée elle peut contenir des sous-tables parallèles. C'est le cas des exemples 2 et 3 cités pour illustrer l'emploi de la directive COND. Les parties parallèles identiques ou non, seront éditées en tables séparées avec des titres différents construits à partir de celui qui est donné dans la directive.

Une table ne peut faire l'objet, à la fois, de directives REG et SEP. On ne peut trouver qu'une seule directive SEP. Une directive SEP annihile l'effet d'une éventuelle directive NUL.

7.6.8 - Directive NUL

Cette directive a pour effet d'effacer des fichiers la table à laquelle elle s'applique. Elle est employée en liaison avec la directive ACT. Ainsi si une action PERFORM P1 fait l'objet d'une directive ACT pour devenir : Traitement de ..., la table P1 peut à un niveau de documentation tel que le détail de ce traitement n'offre plus d'intérêt, ne pas être imprimée. Elle fera l'objet d'une directive NUL qui sera alors la seule directive de la table.

Si la table qui fait l'objet de la directive NUL subit aussi d'autres directives et particulièrement une directive REG, la table est regroupée sur chacune des tables qui l'appellent. Une directive NUL aura pour effet de supprimer l'édition de la table elle-même.

7.6.9 - Directive CHGT

CHGT, nouveau titre.

Cette directive permet de modifier le titre d'une table. Le nouveau titre doit avoir 30 caractères au plus, les blancs sont ignorés. Les directives CHGT et NUL s'excluent mutuellement. La succession des directives suivantes est incorrecte :

```
CHGT, TITRE 1.

REG, T.

NUL.
```

Elle ne conduit pas au regroupement de la table titrée en cause avec changement de titre. Ce résultat est obtenu par les directives :

REG, T, TITRE 1.

NUL.

On trouvera en annexe II des exemples d'emploi des directives NUL et CHGT.

En résumé le tableau suivant indique les exclusions entre directives de structure :

GROUPE		1	2			3	
	directives	AJRS	SEP	REG,T	REG,S	NUL	CHGT
1	AJRS	N					
2	SEP		N	N	N	///	
	REG,T		N	N	N		///
	REG,S		N	N	N		
3	NUL		///			N	///
	CHGT			///		///	N

On ne peut trouver, appliquées à la même table, deux directives pour lesquelles la case d'intersection est hachurée. Si la case d'intersection contient un N seule la première directive du fichier est enregistrée. Les cases blanches correspondent aux combinaisons autorisées sans restriction.

7.7 - MEMENTO D'EMPLOI DE LA CHAINE TABOL

En pratique l'emploi de la chaîne TABOL se résume à la construction de deux JØB. Le premier lit le programme donné et édite les tables de décisions à entrées limitées qui sont construites de façon automatique. Le second lit les directives de transformations des tables et édite des tables de décisions à entrées mixtes.

Les éditions dans les deux cas, s'accompagnent du chargement en mémoire de masse de fichiers qui permettent de rappeler le dernier cycle des tables.

7.7.1 - JØB 1

- Carte JØB.
- 6 cartes de créations des fichiers temporaires : HASH, DICT, CTES, MASK, BOR, REFE.
- 5 cartes d'assignations des fichiers permanents : TAB, DEF, PARAG, DO, ENT.
- Carte d'assignation du fichier des programmes TABOL.
- Carte d'appel de PD1A.
- Les trois premières divisions du programme donnée.
- Carte fin de fichier ou carte PROCEDURE DIVISION.
- Carte d'appel de PD1B.
- PROCEDURE DIVISION du programme avec sa carte de titre.
- Carte fin de fichier.
- Carte d'appel de PD1C.
- 1 carte d'option
  - si option C ou R :
    - Noms des paragraphes.
    - Carte fin de fichier.
    - Carte d'appel de PD2A.
- Carte d'appel de PD2B.
- Carte d'appel de PD3A.
- Carte d'appel de PD3B.
- Carte fin de JØB.

7.7.2 - JØB 2

- Carte JØB.
- 5 cartes d'assignation des fichiers permanents : TAB, DEF, PARAG, DO, ENT.
- 5 cartes de création des fichiers temporaires : MASK, INREF, REFE, ANT, FIDØM.
- Carte d'assignation du fichier des programmes TABØL.
- Carte d'appel de PD3C.
- Fichier des directives.
- Carte fin du fichier.
- Carte d'appel de PD3D.
- Carte d'appel de PD3A.
- Carte d'appel de PD3B.
- Carte fin de JØB.

CHAPITRE 8

PRÉSENTATION PRATIQUE DES PROGRAMMES DE LA CHAINE

8.1 - INTRODUCTION

Ces chapitres n'ont pas été écrits dans l'intention de permettre la maintenance des programmes qu'ils décrivent. Ceux-ci ne sont pas industriellement utilisables dans la version présentée ici. Nous nous attacherons à montrer qu'éditer les tables de décisions d'un programme écrit en COBOL est faisable, en présentant les aspects originaux de la réalisation, ceux qu'il faudra retrouver dans une version industrielle, sans détailler les moyens informatiques employés. Une description exhaustive ne peut avoir sa place dans ce texte.

La chaîne que nous décrivons comprend neuf programmes que l'on peut grouper par deux, par référence à leur objet. La première phase est l'analyse syntaxique du programme donné. PD1A analyse la DATA DIVISION, PD1B la PROCEDURE DIVISION. Ces deux programmes construisent les fichiers nécessaires à la recherche des ruptures sémantiques, le fichier des souches actions et conditions, enfin le masque du programme qui est une première approche du codage des tables de décisions.

Ensuite vient le codage des tables "atomiques" (une table par phrase COBOL), l'étude du graphe du programme donné (une table de décisions par paragraphe), la détermination des entrées de circuits et noeuds de losanges et le regroupement des tables. Ceci est réalisé par les deux programmes PD1C et PD2A.

La construction des tables de décisions se termine par la recherche systématique des ruptures sémantiques de tables, réalisée par le programme PD2B.

Le module d'édition des tables de décisions est utilisé indifféremment pour l'édition des tables de décisions à entrées limitées et des tables à entrées étendues. Ce module est lui-même formé de deux programmes : PD3A recherche les instructions dont le texte a plusieurs occurences dans une même table

et construit l'ordre d'édition des lignes des souches. PD3B réalise la mise en page et l'édition de chaque table.

Enfin la construction des différents niveaux des tables de décisions à entrées mixtes sous le contrôle de directives fournies par l'utilisateur fait l'objet des programmes PD3C et PD3D. Le premier lit toutes les directives et traite les directives relatives à la sémantique de chaque table. Le second reconstruit la répartition du programme entre les différentes tables.

La chaîne de programmes que nous venons de présenter peut être figurée par le schéma de la fig. 8.2. Seuls les fichiers principaux ont été représentés encadrés en pointillés. Les flèches simples symbolisent le transfert d'information, les flèches doubles le passage du contrôle entre les programmes. Ceux-ci n'échangent d'informations que par le seul moyen de fichiers. L'organisation des fichiers sera décrite dans ce chapitre à l'occasion de la création de chacun d'eux. La chaîne de programmes est écrite entièrement en COBOL. Il en existe deux versions l'une pour I.B.M. 1130, l'autre pour UNIVAC 1108. Des exemples d'applications sont fournis en annexe.

Le premier chapitre détaille chacun des programmes. On trouvera dans le corps du chapitre les tables de décisions qui ont servi à l'analyse des modules les plus originaux. Chaque programme a fait l'objet d'une documentation, au moyen de la chaîne elle-même. Le volume des documents ainsi édités est tel qu'il a été impossible de les joindre à ce texte. La chaîne TABOL entière occupe en effet plus de 6 500 cartes COBOL.

8.2 - PRESENTATION DE LA PHASE 1

Les trois programmes de cette phase étudient et analysent le programme COBOL donné. Successivement :

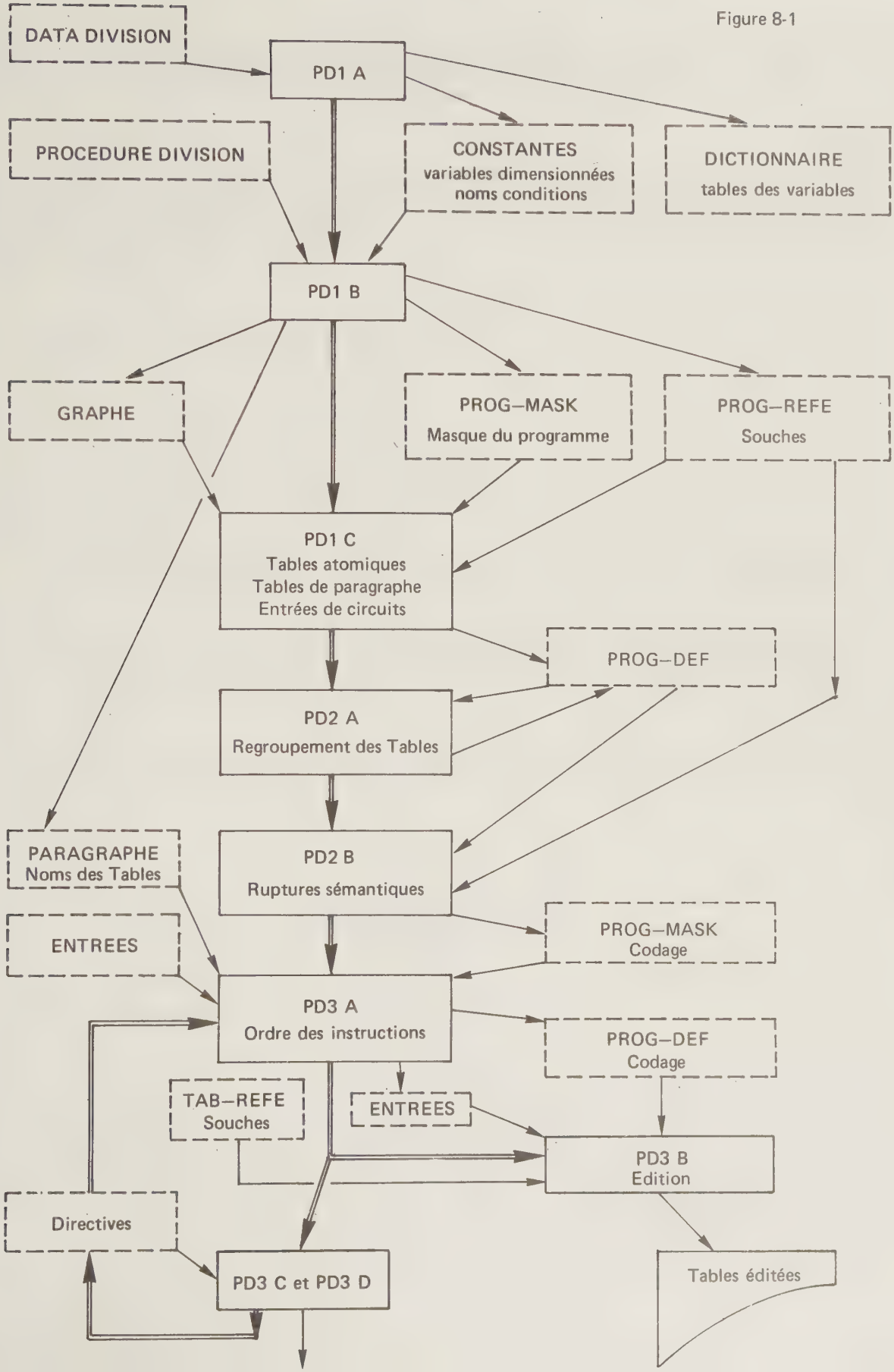
PD1A - Analyse syntaxique de la DATA DIVISION.

PD1B - Analyse syntaxique de la PROCEDURE DIVISION.

PD1C - Construction des tables atomiques et étude du graphe formé par les paragraphes du programme.

Chacune de ces étapes permet non seulement de préparer l'étape immédiatement suivante mais encore prépare l'évaluation des critères de ruptures ou de regroupement de tables qui seront mis en jeu dans la phase suivante. Ainsi la phase 1 A conduit à la construction des tables qui permettront la recherche des ruptures sémantiques et la phase 1 C caractèrise les noeuds de losanges et les

Figure 8-1


DATA DIVISION
PD1 A
PROCEDURE DIVISION
CONSTANTES
variables dimensionnées
noms conditions
DICTIONNAIRE
tables des variables
PD1 B
GRAPHE
PROG–MASK
Masque du programme
PROG–REFE
Souches
PD1 C
Tables atomiques
Tables de paragraphe
Entrées de circuits
PROG–DEF
PD2 A
Regroupement des Tables
PARAGRAPHE
Noms des Tables
PD2 B
Ruptures sémantiques
ENTREES
PROG–MASK
Codage
PD3 A
Ordre des instructions
PROG–DEF
Codage
TAB–REFE
Souches
ENTREES
PD3 B
Edition
Directives
PD3 C et PD3 D
Tables éditées

entrées de circuits point de regroupement ou de séparation des tables. Enfin la phase 1 B édite le fichier des textes d'instructions qui sera utilisé dans les souches actions et conditions des tables de décisions à entrées limitées.

Signalons que le programme lu l'est sur cartes dans la version présentée ici. Nous décrirons successivement chacune des trois étapes de la phase I, en mettant l'accent sur la définition des fichiers qui contiennent le résultat de l'exécution de chacune, bien plus que sur les algorithmes mis en jeu pour obtenir ces résultats.

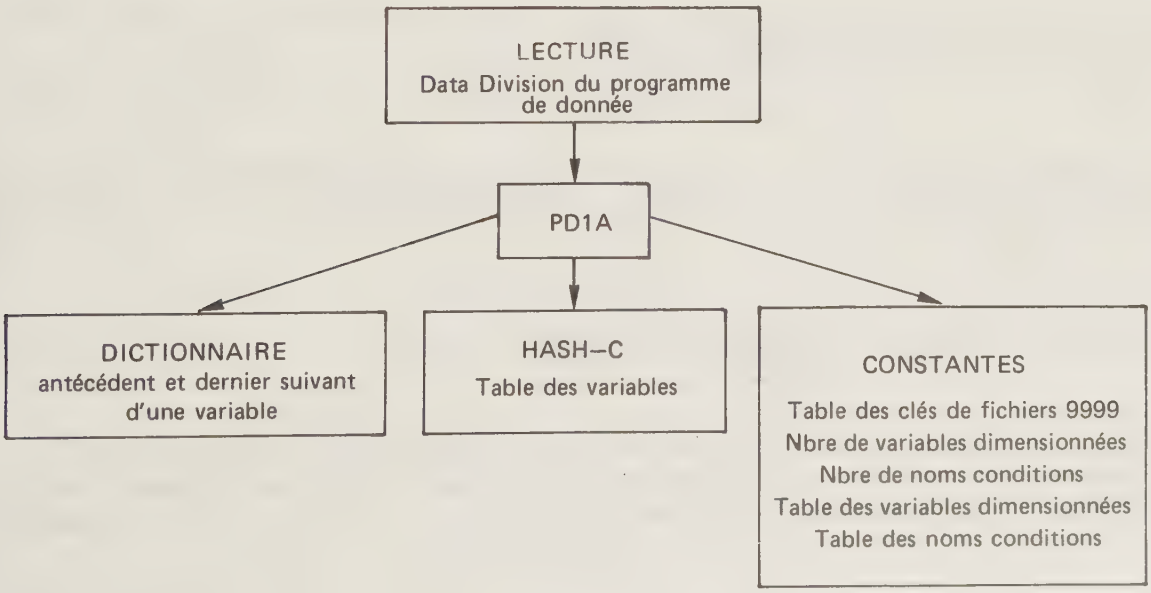
8.2.1 - Compilation de la DATA DIVISION

8.2.1.1 - Présentation générale

L'étude de la DATA DIVISION a deux rôles :

- Préparer les tables nécessaires à la compilation de la PROCEDURE DIVISION. Ces tables sont contenues dans le fichier CONSTANTES.
- Préparer les tables qui permettront dans la phase 2 d'évaluer le critère de rupture de table lié à la sémantique du programme. Ces tables de variables sont éditées dans les fichiers DICTIONNAIRE EN HASH-C.

Le fonctionnement du programme PD1A se résume dans le schéma :



8.2.1.2 - Fichier CONSTANTES. Table des clés de fichiers

La table des clés de fichiers est construite à la lecture de l'ENVIRONMENT DIVISION. Chaque article est formé d'un nom de fichier ou d'enregistrement et du nom de la clé de lecture ou d'écriture correspondante, accessoirement du nombre de lettres de la clé. La première partie du fichier CONSTANTES contient cette table. Une séparation est faite avec les tables suivantes au moyen de 3 enregistrements. Le premier contient des blancs et le nombre 9999, le second le nombre de variables dimensionnées, le troisième le nombre de nom-condition utilisé dans le programme donné. Ces nombres sont utilisés pour lire la suite du fichier.

8.2.1.3 - Fichier CONSTANTES. Table des variables dimensionnées

Le fichier CONSTANTES contient ensuite les noms des variables dimensionnées. Deux variables sont enregistrées dans chaque article. Cette table servira dans la phase d'étude de la PROCEDURE DIVISION à identifier les variables dimensionnées et à séparer leurs indices des virgules et parenthèses qui les encadrent.

8.2.1.4 - Fichier CONSTANTES. Tables des noms-conditions

Les noms-conditions obéissent à une description de la forme :

niveau nom-variable.

88 nom-condition VALUE littéral.

La condition formée du nom-condition sera remplacée dans la phase d'étude de la PROCEDURE DIVISION par la condition :

nom-variable = littéral

jugée plus précise. Pour cela la troisième partie du fichier CONSTANTES contient le nom de la variable, le nom condition et un pointeur d'ordre dans la table suivante des littéraux.

8.2.1.5 - Fichier CONSTANTES. Tables des littéraux

Deux sortes de littéraux sont mémorisées dans cette dernière partie du fichier CONSTANTES. D'une part les littéraux des noms conditions d'autre part les constantes d'une éventuelle CONSTANT SECTION. Les littéraux sont tronqués

à 60 caractères. Cette longueur suffira dans la plupart des cas pour identifier chaque valeur,

Le fichier qui vient d'être décrit de même que les deux suivants sont affectés à une mémoire de masse. Ils sont à accès séquentiel et l'enregistrement est de longueur fixe.

8,2.1.6 - Fichier HASH-C. Tables des variables

Les noms des variables y sont stockés à l'aide d'un adressage dispersé ou hash-coding (d'où le nom du fichier). Pratiquement le fichier HASH-C recopie la table d'adressage dispersé qui est en mémoire centrale pendant le traitement de la DATA DIVISION. Chaque enregistrement contient donc un nom de variable ou un blanc suivi d'une adresse qui est un pointeur, lié à la variable dans les tables suivantes. 401 noms de variables différents peuvent être mis en mémoire. Les qualifications de noms ne sont pas traitées.

8,2,1,7 - Fichier DICTIONNAIRE. Table des arborescences

A l'adresse d'une variable V1 on trouve dans ces deux tables d'une part l'adresse de son précédent direct V0, c'est-à-dire de la variable de groupe de niveau immédiatement inférieur dont elle dépend. On trouve d'autre part l'adresse du dernier "suivant V_n". Ainsi toutes les variables qui sont décrites sous la variable V_1 de groupe ont des adresses en séquence. Ces deux tables forment un dictionnaire des précédents et un dictionnaire chaîné des suivants des graphes arborescents formés par les descriptions de la DATA DIVISION. Les noms de fichiers sont traités comme des variables de niveau zéro.

Pour expliciter le fonctionnement du chaînage autorisé par l'organisation de ces tables nous allons détailler l'exemple de description suivant :

```
1     FD LECTURE ....
2              01 LIGNE.
                     02 FILLER PIC X.
3                    02 W1.
4                          03 W2 PIC 9(2).
5                          03 W3 OCCURS 10.
6                                 04 W4 PICTURE X(2).
7                                 04 W5 PICTURE 9.
                                   04 FILLER PICTURE X.
8                    02 W6.
9                          03 W7.
10                                04 W8 PICTURE 9.
11                                04 W9 PICTURE X.
12                         03 W10 PICTURE X(76).
```

Un tel arbre est traité comme suit :

La table-0 contient le nom des variables, la table-1 parallèle permet de pointer vers les tables suivantes.

TABLE-0	TABLE-1
LECTURE	1
W9	11
W6	8
W4	6
W2	4
W5	7
LIGNE	2
W1	3
W3	5
W10	12
W7	9
W8	10

Les tables 2 et 3 contiennent les derniers suivants et précédent de chaque variable. La troisième colonne rappelle ici le nom de la variable associée.

TABLE-2	TABLE-3	
0	12	LECTURE
1	12	LIGNE
2	7	W1
3	0	W2
3	7	W3
5	0	W4
5	0	W5
3	12	W6
8	11	W7
9	0	W8
9	0	W9
8	0	W10

TABLES DE DOCUMENTATION.

Table 1 RECHERCHE-MOT

Construction d'un mot

Environment division

LE CARACTERE LU EST UN ESPACE	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	O	Ø	Ø	N
UN MOT EST CONSTRUIT	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	O	Ø	N	
LE MOT EST FACULTATIF	Ø	N	N	N	N	N	N	N	N	N	N	N		
LA DIVISION EN COURS EST LA		1	1	2	2	2	2	2	2	2	9	3		
LE MOT EST SELECT				Ø	N	N	N	N	N	N	N			
LE MOT EST UN NOM DE FICHIER					Ø	N	N	N	N	N	N			
LE MOT EST ACTUAL OU SYMBOLIC						Ø	N	N	N	N	N			
LE MOT SE TERMINE PAR UN POINT							Ø	Ø	N	N	N			
LE MOT EST UN NOM DE CLE							Ø	N	O	N	N			
LE MOT EST UN TITRE DE DIVISION		O	N							O	N			
METTRE LE CARACTERE EN MEMOIRE														*
LE MOT SUIVANT SERA UN NOM DE FICHIER				*										
METTRE LE NOM EN MEMOIRE					*									
LE MOT SUIVANT SERA UN NOM DE CLE						*								
SUPPRIMER LE POINT							*							
CONSTRUIRE L'ENREGISTREMENT DE CONSTANTES FICHIER-CLE							*		*					
RAPPELER LA NOUVELLE DIVISION		*								*				
REMETTRE A BLANC LE MOT	*	*	*	*	*	*	*	*	*	*	*			
LIRE LE CARACTERE SUIVANT	*			*	*	*							*	*
LIRE LA CARTE SUIVANTE		*	*				*	*	*	*	*			
GO TO RECHERCHE-MOT	*	*	*							*	*		*	*
VERS TABLE SUIVANTE												*		

Table 2 R3 Mots de la DATA DIVISION

LE MOT INDIQUE UN TITRE DE SECTION	Ø	N	N	N	N	N	N	N	N	N	N	N	N	N	N
LE MOT INDIQUE LE TITRE DE LA PROCEDURE		Ø	N	N	N	N	N	N	N	N	N	N	N	N	N
LE MOT INDIQUE UN NUMERO DE NIVEAU OU FD DANS LA FILE SECTION			Ø	N	N	N	N	N	N	N	N	N	N	N	N
LE MOT EST UN NOM DE VARIABLE				Ø	Ø	Ø	Ø	Ø	Ø	Ø	N	N	N	N	N
LE MOT EST UN NOM CONDITION				Ø	N	N	N	N	N	N					
LA VARIABLE EST UN NOM D'ENREGISTREMENT D'UN FICHIER A ACCES ALEATOIRE					Ø	Ø	N	N	N	N					
LE NIVEAU DE LA VARIABLE INDIQUE UN DEBUT D'ARBRE, FD DANS LA FILE SECTION 01 OU 77 DANS LES AUTRES							Ø	Ø	N	N					
LE MOT SE TERMINE PAR UN POINT					Ø	N	Ø	N	Ø	N	Ø	Ø	Ø	N	N
LE MOT EST OCCURS											Ø	N	N		
UN ANTECEDENT DE LA VARIABLE EST DIMENSIONNE												Ø	N		
LE MOT EST VALUE ET LA VARIABLE EST UN NOM CONDITION OU EST DANS LA CONSTANTE														Ø	N
RAPPELER DANS JO LA SECTION EN COURS D'ETUDE	*														
RAPPELER LE NIVEAU LE MOT SUIVANT SERA UN NOM DE VARIABLE J1 = 2			*												
METTRE DANS LA TABLE DES NOMS CONDITION LE NOM DE LA VARIABLE ET DE SON ANTECE–DENT				*											
CONSTRUIRE LE POINTEUR DE LA TABLE-1 ET L'ADRESSE DU NOM DE LA VARIABLE				*	*	*	*	*	*	*					
CREER L'ENREGISTREMENT DE CONSTANTES					*	*									
CONSTRUIRE LE LITTERAL DE LA VALEUR														*	
IL N'Y A PAS D'ANTECEDENT							*	*							
CHERCHER L'ANTECEDENT ; LUI ASSOCIER CE SUIVANT									*	*					
AJOUTER LE NOM DE LA VARIABLE A LA TABLE DES VARIABLES DIMENSIONNEES											*	*			
INITIALISER LA DESCRIPTION D'UNE VARIABLE					*		*		*			*	*	*	
RECOPIER LES TABLES DANS LES FICHIERS		*													
EFFACER LE MOT	*		*	*	*	*	*	*	*	*	*	*	*	*	*
LIRE LE CARACTERE SUIVANT			*	*		*		*		*	*				*
LIRE LA CARTE SUIVANTE	*				*		*		*			*	*	*	
GO TO RECHERCHE–MOT	*		*	*	*	*	*	*	*	*	*	*	*	*	*
FIN		*													

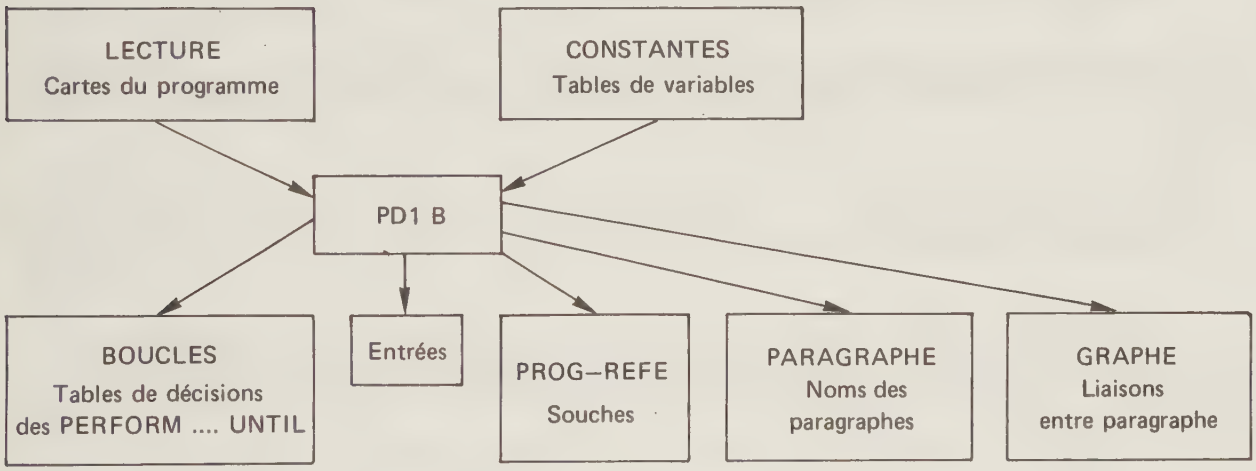
8.2.2 - Analyse syntaxique de la PROCEDURE DIVISION

8.2.2.1 - But du programme

Ce programme réalise un reconnaisseur syntaxique partiel de COBOL. Seuls le verbe PERFORM et les conditions, y compris les actions conditionnelles ON SIZE ERROR AT END INVALID KEY, sont analysés en détail. Nous indiquons dans chaque cas le traitement réalisé.

Le résultat de cette analyse est contenu dans deux fichiers. L'un PROG-REFE contient le texte de chacune des instructions ; conditions et actions. L'autre PROG-MASK mémorise sous la forme des clés d'accès au fichier précédent la séquence des instructions. Ces fichiers seront utilisés dans la phase suivante pour construire les tables de décisions de paragraphes et évaluer les critères de regroupement des tables.

Le programme admet 6 fichiers de sortie :



L'analyse syntaxique est organisée autour de 3 modules principaux, de façon descendante.

1) Recherche, construction et reconnaissance d'un mot du programme, mots clés, littéraux, identificateurs, identificateurs dimensionnés et leurs indices.

2) Traitement des instructions. A chaque verbe COBOL correspond un groupe de paragraphes qui traite l'instruction.

3) Procédures utilitaires communes à ces modules. Nous citerons : Enregistrer une instruction de souche dans le fichier PROG-REFE. Créer le masque d'une phrase conditionnelle dans le fichier PROG-MASK. Construire un littéral alphanumérique. Reconnaître un littéral numérique, etc.

Nous étudierons ici la structure des fichiers de sorties.

8.2.2.2 - PROG-REFE

Le fichier PROG-REFE contient le texte des souches conditions et actions ou ruptures de séquence. Ce fichier est écrit séquentiellement et sera utilisé par la suite en accès aléatoire. Les textes d'instructions ont 95 caractères au maximum. Le verbe de l'instruction est séparé du texte. L'enregistrement contient encore le nombre de caractères de l'instruction et la clé d'accès à l'enregistrement d'entrées associées. Dans la version décrite ici, les enregistrements sont de longueur fixe.

Les conditions se caractèrisent par une zone verbe blanche.

Les instructions que l'on trouvent dans ce fichier sont des instructions relevées dans le programme COBOL, ou des instructions générées à partir de celles-ci.

8,2,2.3 - Entrées

Le fichier ENTREES contient un enregistrement pour chaque instruction mise dans PROG-REFE. Pour une action ou une rupture de séquence l'enregistrement contient la seule entrée * qui puisse être éditée. Pour une condition l'enregistrement contient les deux entrées O et N.

Le fichier entrée sera utilisé lors de l'édition des tables sans subir aucune modification. Il interviendra ensuite dans la création des tables à entrées étendues.

8.2,2.4 - PROG-MASK

C'est une succession de nombres formés :

- d'une part d'un pointeur,

- d'autre part d'une clé d'accès au fichier des souches.

La forme codée ainsi éditée n'est pas celle d'une table de décisions telle qu'elle a été définie en 4.2.4, mais un masque qui traduit la structure d'une phrase COBOL.

Le pointeur peut être 7, Il indique alors que l'instruction dont la référence suit est une condition.

8. L'instruction dont la clé suit est une action ou une rupture de séquence. C'est la première d'une séquence d'actions.

9. L'instruction référencée est une action ou une rupture de séquence. Elle s'exécute en séquence avec la précédente.

0. Le pointeur 0 indique les parenthèses de construction d'une phrase conditionnelle. Le nombre qui suit peut être :

0 début et fin de phrase.

1 équivalent de IF.

2 équivalent de THEN. Ce terme est introduit systématiquement s'il n'est pas présent dans la phrase COBOL.

3 équivalent de ELSE. Le nombre de IF et de ELSE n'est pas équilibré au cours de l'analyse de la phrase.

4 et 5. Le nombre qui suit n'a aucune signification. Il vaut zéro. Pour expliquer la présence de ces pointeurs il nous faut étudier le traitement que subissent les conditions composées avec les conjonctions OR et AND et la négation NOT, appliquées à des conditions simples,

8.2.2.4.1 - Traitement des conditions composées. Résultats.

Les textes de chaque condition simple sont référencés dans le fichier PROG-REFE, parallèlement la liste des références et la présence éventuelle de conjonctions sont mémorisées. L'expression logique ainsi constituée est postfixée. A la fin de la lecture du texte, la forme postfixée est reprise et transformée en un masque qui sera édité dans le fichier PROG-MASK.

- Ainsi I1 NOT EQUAL TO 1 deviendra :

dans PROG-REFE : 1824 .,.....I1 EQUAL TO 1

et dans PROG-MASK : 7 1824 4 0000 5 0000

Ce qui s'énonce : Si la condition I1 = 1 est satisfaite, la condition composée n'est pas satisfaite (4) sinon elle l'est (5).

- Ainsi TOTAL LESS 100 OR COMMANDE GREATER 30000

sera traité comme :

Dans le fichier PROG-REFE : 1825 .. TOTAL LESS 100

1826 .. COMMANDE GREATER 30000

Et dans PROG-MASK la suite de nombres :

7 1826 5 0000 7 1825 5 0000 4 0000

Nous écrirons par la suite en abrégé : 7C2 0 7C1 0 N équivalent à C1 OR C2 ce qui peut s'exprimer par une phrase comme :

Si C2 est vérifié la condition composée est vérifiée sinon, si C1 est vérifié la condition est encore vérifiée, sinon (ni C1 ni C2 ne sont vrais) alors la condition composée n'est pas vérifiée.

- De même C1 and C2 sera traduit par le masque : 7C1 7C2 0 N N

Toutes les formes de conditions composées admises en COBOL E sont acceptées et traitées suivant ce schéma.

8.2.2.4.2 - Traitement des conditions composées. Algorithmes.

La postfixation de l'expression logique est conduite au cours de la séparation des conditions simples. L'algorithme mis en jeu est classique. Nous ne le détaillerons pas pour fournir simplement la table de décisions qui gère la lecture des conditions. Elle est activée pour chaque mot de la condition composée.

Table 1 P-CON Séparation des conditions simples

LE MOT EST FORME DE PN1	Ø	N	N	N	N	N	N	N	N	N	N	N	N
LE MOT EST UN NOM CONDITION		Ø	N	N	N	N	N	N	N	N	N	N	N
NOT			Ø	Ø	N	N	N	N	N	N	N	N	N
OR					Ø	Ø	N	N	N	N	N	N	N
AND							Ø	Ø	N	N	N	N	N
UNEQUAL									Ø	Ø	N	N	N
UN ADJECTIF DE CONDITION											Ø	N	N
ZERO ET IL N'Y A PAS EU D'ADJECTIF DANS LA CONDITION												Ø	N

SOUS-TABLE POST-F

LE TERME DU SOMMET DE LA PILE DES OPERATEURS EST DE PRIORITE INFERIEURE			Ø	N	Ø	N	Ø	N	Ø	N			
EMPILER DANS LA PILE DES OPERATEURS PN1 (: CODE = 1	*												
GENERER L'INSTRUCTION EQUIVALENTE		*											
GENERER NOT									*	*			
AFFECTER UNE REFERENCE AU TEXTE									*	*	*	*	
EMPILER DANS LA PILE DU RESULTAT 70000 + REFERENCE		*							*	*	*	*	
GENERER EQUAL DANS LE TEXTE									*	*			
CONSTRUIRE LE TEXTE DE LA CONDITION SIMPLE											*	*	*
CREER PN2					*	*	*	*					
STOCKER LE TEXTE DE LA CONDITION					*	*	*	*					

SOUS-TABLE POST-F

EMPILER L'OPERATEUR NOT = 9			*										
OR = 7					*								
AND = 8							*						
RECOPIER LE SOMMET DE LA PILE OPERATEUR AU SOMMET DE LA PILE DES RESULTATS				*		*		*					
GO TO POST-F				*		*		*					
GO TO P-C30		*			*		*		*	*	*	*	*
RETOUR (CONSTRUCTION DU MOT SUIVANT)	*		*										

Table 2 — P-C30	Parenthèses droites			
LE MOT CONTIENT PN1)	Ø	Ø	Ø	N
LE SOMMET DE LA PILE DES OPERATEURS INDIQUE UNE (	Ø	N	N	
L'ADJECTIF DE LA CONDITION SIMPLE EN COURS A ETE LU		Ø	N	
RAPPELER DANS PN2 LE NOMBRE PN1			*	
DEPILER LES OPERATEURS JUSQU'A UNE (LES RECOPIER DANS LA PILE DU RESULTAT		*		
SUPPRIMER LE SOMMET DE LA PILE DES OPERATEURS	*	*		
SOUSTRAIRE 1 DE PN1	*	*		
INSCRIRE UNE) DANS LE TEXTE DE LA CONDITION	*	*		
GO TO P-C30	*	*		
RETOUR (CONSTRUCTION DU MOT SUIVANT)			*	*

Les sujets et adjectifs sous-entendus sont admis. On voit que l'algorithme proposé ne distingue pas les parenthèses qui créent la structure logique de la condition proposée des parenthèses trouvées dans le texte de chaque condition simple : indices et expressions arithmétiques. Ces parenthèses supplémentaires s'annulent deux à deux ensuite.

La forme postfixée ainsi construite est ensuite analysée de façon à créer le masque de la condition composée. Celui-ci est une forme préfixée d'expression dans laquelle les références aux conditions simples sont des opérateurs binaires, et les branches pendantes Oui et Non sont les opérandes. C'est la forme codée de l'arborescence binaire équivalente à la condition composée. Les branches pour lesquelles la condition est satisfaite sont repérées par le pointeur 5, celles pour lesquelles elle n'est pas satisfaite par le pointeur 4.

a) La forme préfixée est lue de droite à gauche. Le premier opérateur OR ou AND est trouvé, Puis à partir de sa position, et vers la gauche, on recherche la première référence de condition. Il peut se trouver, entre les deux, des occurences de NOT. Le nombre d'inversions NOT est compté. La zone ainsi décrite est effacée et le masque élémentaire formé de la référence à la condition et de 5 puis 4, ou 4 puis 5, suivant la parité du nombre d'inversion, est construit,

b) La recherche de l'autre opérande s'accompagne également d'une recherche des inversions NOT. La variable LL1 vaut d'abord 5 et LL2 vaut 4. A la rencontre d'un NOT, les deux valeurs LL1 et LL2 sont inversées. Le masque élémentaire du second opérande est ainsi dans tous les cas :

C · LL1 LL2.

Soit : C 5 0000 4 0000 si le nombre d'inversions est pair

ou : C 4 0000 5 0000 sinon,

c) Le masque contient la forme en cours de construction. L'opérateur et le deuxième opérande sont repérés dans la forme préfixée. Le nouveau masque peut être généré. La variable LLR permet de distinguer la valeur du pointeur qui doit être remplacée, dans le masque, par l'opérande de la pile.

Ainsi soit C1 4 C2 5 4 le contenu du masque, C3 la condition et l'opérateur OR. Les trois termes :

C3 LL1 LL2 doivent être recopiés sur tous les termes 4 du masque et LLR vaut 4. Dans le cas de l'opérateur AND LLR vaut 5.

d) L'opérateur suivant est ensuite recherché vers la droite dans la forme préfixée, Les éventuelles occurences de NOT rencontrées conduisent à échanger les 4 et les 5 dans le masque, Trois cas peuvent se présenter :

- l'opérateur suit immédiatement. Le traitement est repris en b.

- avant l'opérateur suivant, il est trouvé une condition. Les deux opérateurs OR et AND étant commutatifs, le traitement standard peut également être repris (en c).

- Deux références de conditions, ou d'avantage séparent l'opérateur suivant. Le contenu du masque doit alors être mis dans un buffer annexe. L'adresse de ce buffer est rappelée dans la forme préfixée, et le traitement est repris en a.

Deux limitations doivent être signalées. La première tient à la taille prévue dans le programme, pour la zone où le masque est construit. Le diagnostic :

EXPRESSION LOGIQUE TROP LONGUE

signale un dépassement. La seconde est liée à la présence de buffers annexes. Un seul buffer est prévu. S'il apparait qu'un second est nécessaire à l'étude d'une expression le diagnostic :

EXPRESSION LOGIQUE COMPLEXE

est édité,

C'est la présence de conditions composées et la complexité de l'algorithme qui les étudie qui a interdit de générer directement le codage des tables atomiques et imposé le passage par l'intermédiaire du masque.

8.2.2.4.3 - Instructions conditionnelles

Nous avons donné ce nom générique aux instructions générées à l'occasion des mots réservés COBOL :

AT END ...

ON SIZE ERROR ...

INVALID KEY ...

DEPENDING ON ...

Chacune de ces 4 formes conduit à la génération d'instructions qui expriment en mots anglais, les conditions positives qui s'y rattachent, et à l'édition d'un masque de phrase conditionnelle sous la forme générale décrite ci-dessus. Nous détaillerons rapidement sur un exemple chacun de ces cas.

8.2,2.4.3.1 - READ LECTURE AT END GO TO FIN

conduit à la génération dans PROG-REFE des instructions :

```
1432   READ LECTURE RECORD
1433   ONE MORE RECORD IN LECTURE
1434   GO TO FIN
```

et dans PROG-MASK au masque :

```
0 0000  0 0001  7 1433  5 0000  4 0000  0 0002  8 1432  0 0003  8 1434  0 0000
```

*8.2.2.4.3.2 - COMPUTE I = K * L ON SIZE ERROR GO TO ER-HC*

est traduit par les instructions :

```
1435   COMPUTE I = K * L
1436   I-SIZE GET K * L
1437   GO TO ER-HC
```

et le masque :

```
0 0000  0 0001  7 1436  5 0000  4 0000  0 0002  8 1435  0 0003  8 1437  0 0000
```

8.2.2.4.3.3 - WRITE LIGNE INVALID KEY DISPLAY 'BOF'

est traité de façon différente selon que le fichier dont LIGNE est un enregistrement est séquentiel ou non. (Signalons qu'en COBOL I.B.M. 1130 l'écriture sur un fichier affecté à un disque se fait obligatoirement avec la clause INVALID KEY quelque soit le mode d'accès.)

Si le fichier de LIGNE est à accès aléatoire, nous appelerons CLE la variable qui contient la clé d'accès. L'exemple donné devient alors :

```
1438   WRITELIGNE
1439   CLE VALID KEY
1440   DISPLAY HO5H 'BOF'
```

et le masque traduit la structure logique naturelle :

0 0000 0 0001 7 1439 5 0000 4 0000 0 0002 8 1438 0 0003 8 1440 0 0000

Si le fichier est à accès séquentiel, la condition générée est de la forme : 1439 MORE STILL PLACE IN nom du fichier.

8.2.2.4.3.4 - GO TO P1 P2 P3 DEPENDING ON J

Une telle phrase est décomposée en une phrase équivalente de la forme :

IF J = 1 GO TO P1 ELSE IF J = 2 GO TO P2 ELSE IF J = 3 GO TO P3 ELSE EXIT EN ERREUR,

Soit dans PROG-REFE :

```
1441  J = 1
1442  GO TO P1
1443  J = 2
1444  GO TO P2
1445  J = 3
1446  GO TO P3
```

L'instruction EXIT EN ERREUR est générée au début de l'étude de la PROCEDURE DIVISION, Le numéro de l'enregistrement de PROG-REFE est 2. Le masque équivalent construit est :

0 0000 0 0001 7 1441 0 0002 8 1442 0 0003 0 0001 7 1443 0 0002 8 1444

0 0003 0 0001 7 1445 0 0002 8 1446 0 0003 8 0002 0 0000

Le traitement que subissent les instructions conditionnelles est organisé de façon à autoriser la présence de telles instructions dans une phrase conditionnelle IF ... THEN ... ELSE ...

8.2.2.5 - Fichier BOUCLES

Le verbe PERFORM est le seul verbe qui conduise à une analyse syntaxique détaillée. Chaque verbe PERFORM complet (avec le clause UNTIL) impose l'édition d'une table de décisions séparée. Elle regroupe les conditions qui suivent UNTIL.

Ainsi la phrase :

PERFORM P1 THRU P2 VARYING I FROM I1 BY I2 UNTIL I GREATER I3 OR LETTRE (I) NOT EQUAL SPACE
AFTER J FROM J1 BY J2 UNTIL J LESS J3 AND P (I, J) = ZERO.

sera représentée par la table suivante ;

			P1$	1250 $	P2
I GREATER I3	Ø	N	N	N	N
LETTRE (I) = SPACE		Ø	Ø	Ø	N
J LESS J3		Ø	Ø	N	
P (I, J) = ZERO		Ø	N		
ADD I2 TO I		*			
MOVE J1 TO J		*			
PERFORM P1 THRU P2			*	*	
ADD J2 TO J			*	*	
GO TO P1$1250$P2		*	*	*	
EXIT P2	*				*

Le codage de cette table atomique est directement construit à partir de la forme postfixée des conditions qui suivent les UNTIL. Le codage est écrit dans le fichier BOUCLES sous la forme générale du codage des tables de décisions à entrées étendues. Il ne subira aucun traitement avant l'édition à la phase 3.

Dans le texte principal du masque, édité dans PROG-Mask, on trouvera référence aux textes des différentes instructions d'initialisation des indices de description I et J et une référence à une instruction de la forme :

PERFORM P1$1250$P2

d'appel à la table fermée dessinée ci-dessus.

8.2,2.6 - Fichier PARAGRAPHE

Fichier à accès aléatoire direct, ce fichier contient la table à adressage dispersé, des noms des paragraphes du programme lu. Ce fichier sera utilisé dans toutes les étapes suivantes. Il permet la liaison entre le nom d'un paragraphe et la position du codage des instructions. Il sera enfin utilisé lors de l'édition des tables ; il contient le titre de chaque table éditée.

La présence d'un titre de paragraphe ou de section est mémorisée dans le masque du programme par un enregistrement pour lequel le pointeur vaut 1 ; le nombre qui le suit est la clé d'accès au nom du paragraphe dans le fichier PARAGRAPHE. 203 noms de paragraphes ou de sections différents peuvent être mis en mémoire dans ce fichier. La présence de deux paragraphes de même nom dans deux sections différentes (synomymes) n'est pas autorisé.

8.2.2.7 - Fichier GRAPHE

C'est un fichier à accès séquentiel. Il permet de construire une ébauche du dictionnaire des suivants du graphe formé par les paragraphes du programme, Il ne fait référence qu'aux liaisons explicites entre paragraphes, créées par des instructions :

GO TO ou PERFORM,

à l'exclusion des liaisons de paragraphes en séquence. Chaque enregistrement du fichier GRAPHE est composé d'un nom de paragraphe et d'un chiffre. Ce chiffre peut être :

0, Le nom du paragraphe est alors l'extrémité initiale de toutes les liaisons indiquées ensuite, jusqu'au prochain enregistrement contenant un zéro.

1, La liaison vers le paragraphe se fait par un verbe PERFORM. L'option THRU est utilisée.

2. Le nom du paragraphe précisé dans l'enregistrement correspond à celui qui suit l'option THRU. Ainsi une instruction PERFORM P1 THRU P2 conduit à l'édition de deux enregistrements consécutifs :

1 P1

et

2 P2

3. La liaison avec le paragraphe indiqué se fait par PERFORM. L'option THRU n'est pas utilisée.

4. La liaison est créée par une instruction GO TO.

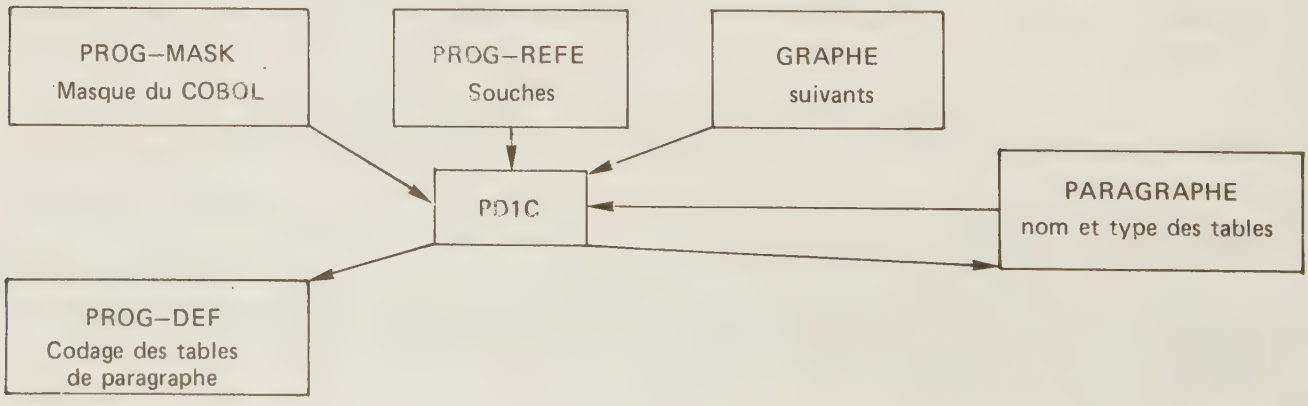
8.2.2.8 - Conclusion

Nous avons décrit les fichiers édités par la passe d'analyse syntaxique PD1B. Nous ne décrirons pas davantage les moyens informatiques et logiques mis en jeu pour obtenir ces résultats. Un tel exposé, fastidieux, serait en dehors de notre propos. Les intermédiaires de programmation ne présentent pas d'originalités particulières ; nous ne prétendons pas avoir réalisé un analyseur particulièrement efficace. Le programme PD1B, écrit en COBOL, est relativement volumineux (1 700 cartes).

8.3 - PROGRAMME PD1C

Cette étape transforme le codage du COBOL réalisé par le programme précédent en un codage de tables de décisions. Elle forme d'abord des tables de décisions atomiques et regroupe les tables atomiques d'un même paragraphe COBOL. Le programme étudie ensuite le graphe formé par les relations logiques entre paragraphes. Le dictionnaire des suivants de ce graphe est construit d'après le fichier GRAPHE, les circuits et entrées de circuits sont recherchés de façon à préparer la construction des tables réunissant plusieurs paragraphes.

Le programme PD1C utilise les fichiers indiqués par le schéma :



8.3.1 - Codage des tables atomiques

8.3.1.1 - Rappels

Le codage du COBOL est lu dans le fichier PROG-MASK. Chaque enregistrement contient un nombre de 5 chiffres. Rappelons que le premier caractèrise l'articulation de la phrase : POINTEUR.

0 pour IF, THEN, ELSE début et fin de phrase.

1 Titre de paragraphe les 4 chiffres suivant sont l'adresse dans le fichier PARAGRAPHE du nom du paragraphe.

2 Référence à l'instruction spéciale EXIT Nom du paragraphe. Celle-ci sera utilisée pour les paragraphes qui sont une fin de module appelé par une instruction PERFORM.

4 Suivant NON d'une condition.

5 Suivant OUI d'une condition.

7 La référence qui suit est celle d'une condition.

8 Première instruction d'une séquence d'actions.

9 Actions ou rupture de séquence.

Le nombre de 4 chiffres qui suit le pointeur forme une clé d'accès dans le fichier des textes de souches PROG-REFE.

8.3.1.2 - Lecture

Le fichier PROG-MASK est lu et recopié dans le tableau :

```
02 TABLE-ATOMIQUE OCCURS 2040.
    03 T PICTURE 9.
    03 TD PICTURE 9(4) COMP-1.
```

Les tables inconditionnelles sont reconnues au cours de la lecture. Elles sont mises en forme par un module particulier et simple. En cours de lecture, la position des IF, THEN, ELSE des tables conditionnelles est mémorisée dans PILE avec l'indice IP.

(* Voir la suite de ce paragraphe page 154/155.)

8.3.1.3 - Construction du codage d'une table atomique

Les phrases qui codent une table atomique du paragraphe en cours d'étude sont écrites à la suite dans le tableau :

02 TABLE-PARAGRAPHE OCCURS 2040.

03 TS PICTURE 9.

03 TDS PICTURE 9(4) COMP-1.

LES PHRASES inconditionnelles sont mises dans ce tableau dès la lecture. L'indice NT du dernier terme de chaque table atomique, est rappelé dans la colonne SUITE.

La transformation d'une phrase conditionnelle commence par l'écriture de termes équivalents à ELSE NEXT SENTENCE de façon à équilibrer le nombre de IF et celui de ELSE. Les termes générés sont 0 0003 2 8888.

L'étude des suivants oui et non de chaque condition est entreprise et les termes trouvés remplacent les occurences des pointeurs 5 et 4 dans la phrase. La mise en mémoire au cours de la lecture de la phrase de la position occupée par les IF THEN et ELSE (tableau PILE) facilite la recherche. Chaque séquence se termine par une référence à une rupture de séquence. La récursivité présentée par les séquences conditionnelles est réduite par l'emploi d'une pile notée MEMO, le sommet en est pointé par P. Le résultat est ainsi construit dans les lignes TS TDS au cours d'une seule description de la phrase. Tables de documentation 2 et 3.

8.3.2 - Regroupement des tables atomiques d'un paragraphe

Il s'agit de regrouper en une seule table de décisions les tables atomiques qui viennent d'être créées. Le tableau SUITE indique la position du début de chaque table. L'appel de chaque table à la suivante est caractérisé par la référence 8888. Le regroupement est entrepris au cours d'une lecture de gauche à droite des lignes TS et TDS. A la rencontre d'un appel à la table suivante :

- La tête de la table appelée est recherchée à l'aide de SUITE.

- L'indice de description est rappelé dans une pile pour reprise de la description de la table d'appel.

- Le pointeur du premier terme de la table appelée est construit en fonction de sa valeur précédente et de celle (2 ou 3) du pointeur de la référence de chaînage.

En fin la description de la table appelée est introduite. Le résultat est édité terme par terme dans le fichier de sortie PROG-DEF. La rencontre d'une fin de table, pointeur et référence nuls, conduit à dépiler le sommet de la pile et reprendre la description du codage à partir de cet indice. Si la pile se trouve vide le regroupement est terminé.

8.3.3 - Etude du graphe du programme

Cette étude est articulée en deux phases.

8.3.3.1 - Construction et structure du dictionnaire des suivants

Rappelons que lors de l'analyse syntaxique du texte COBOL :

- Les noms de paragraphes ont été dispersés dans le fichier PARAGRAPHE par hash-code. L'adresse du nom a été mise en mémoire dans PROG-MASK dans le premier terme du paragraphe avec le pointeur 1.

- Le fichier GRAPHE contient les appels de paragraphe entre eux. Les instructions explicites GO TO et PERFORM entraînent l'édition d'un enregistrement dans GRAPHE, sous la forme suivante :

0 Nom du paragraphe appelant (antécédent)

1 NOM 1 }
2 NOM 2 } Noms des suivants
3 NOM 3 }
4 NOM 4 }

Les pointeurs 1 et 2 sont utilisés si l'appel se fait par PERFORM NOM1 THRU NOM2.

Le pointeur 3 est utilisé pour l'appel PERFORM NOM 3.

Le pointeur 4 pour l'appel GO TO NOM4.

Il faut remarquer que ce fichier ne contient pas mention des appels de paragraphes en séquence. Le dictionnaire des suivants construit à l'aide des fichiers GRAPHE et PARAGRAPHE au début du programme PD1C a une structure schématisée par la figure suivante :

PARAGRAPHE	NOMBRE-SUIVANTS	SUIVANT	TIPE
Noms des paragraphes 203 lignes	nombre de suivants du parag.	38 colonnes, adresses des suivants	1 chiffre

Ainsi si dans le paragraphe P1 on trouve l'instruction	on trouvera sur la ligne P1 et à la première colonne libre : NOMBRE-SUIVANTS
GO TO P2	L'adresse de P2
PERFORM P3	L'adresse de P3 + 3000
PERFORM P4 THRU P5	L'adresse de P4 + 1000 et à la colonne suivante l'adresse de P5 + 2000

Le dictionnaire est construit avant la création des tables atomiques et la lecture du fichier PROG-MASK. Les appels en séquence des paragraphes sont simplement repérés au cours de la phase de regroupement des tables atomiques d'un paragraphe par les références 8888 qui ne peuvent conduire à un regroupement. Ils sont ensuite mentionnés dans le dictionnaire. Table de documentation 4.

La colonne TIPE caractèrise les appels qui sont faits au paragraphe correspondant :

1 pour un paragraphe appelé par PERFORM avec l'option THRU

2 pour un paragraphe sortie de module appelé par PERFORM

3 pour un paragraphe à la fois entrée et sortie de module appelé par PERFORM

4 pour les autres paragraphes.

Cette colonne ainsi initialisée prépare le travail de recherche des entrées de circuits.

8.3.3.2 - Recherche des entrées de circuits

Ce module n'est activé que sur option. Les 3 options possibles sont :

R Le programme recherche les entrées de circuits et lit les têtes de module proposés par l'utilisateur.

C Le programme lit les têtes de module proposées par l'utilisateur. Il doit dans ce cas y en avoir au moins une sur chaque circuit du graphe.

Autre Il n'y aura pas de regroupement des paragraphes. Une table sera éditée pour chaque paragraphe. Le dictionnaire des suivants n'est pas construit.

- Description de l'algorithme utilisé

La recherche des entrées de circuits se fait de façon élémentaire. La description de tous les chemins du graphe s'accompagne de la mise en mémoire dans le tableau PILE des sommets décrits depuis le point d'entrée. Un sommet est déterminé dans le dictionnaire. Le traitement consiste à rechercher dans PILE une éventuelle occurence de ce sommet dans le chemin déjà décrit. S'il en est trouvé une, un circuit est trouvé dans le graphe, et le sommet est une entrée de ce circuit. La variable TIPE associée au paragraphe prend une valeur qui le rappelle :

2 devient 3, 4 devient 1, 1 et 3 restent identiques.

La description du chemin est reprise avec le premier suivant non encore étudié de l'antécédent de l'entrée de circuit trouvée. Le tableau PILE reste identique ; un seul circuit peut en effet admettre plusieurs entrées.

Dans le cas où le nouveau paragraphe étudié n'est pas trouvé dans la suite des paragraphes qui définissent le chemin décrit, son adresse est ajoutée au Tableau PILE. Le traitement est repris avec le premier de ses suivants non encore étudié.

Si le suivant ainsi déterminé dans chaque cas se trouve être atteint

par une instruction PERFORM, c'est-à-dire si le terme du dictionnaire est supérieur à 1 000, l'adresse du paragraphes fin de module est rappelée dans MEMO, et la position dans pile du paragraphe de début de module, dans SUITE.

Le tableau MEMO n'est pas consulté à chaque nouveau paragraphe traité. Seul le dernier suivant d'un paragraphe est recherché dans MEMO. Cette particularité est imposée par la possibilité qu'offre le langage COBOL de sortir d'un module appelé par PERFORM parune simple instruction GO TO adressée à un paragraphe hors du module, et de créer éventuellement ainsi un circuit.

Enfin si tous les suivants d'un paragraphe ont été étudiés, ou s'il n'admet pas de suivants, la description du graphe est reprise à partir de l'antécédent de ce paragraphe sur le chemin ; il est déterminé à l'aide du tableau PILE, qui est ensuite réduit. Si ce paragraphe est l'entrée du programme, c'est-à-dire s'il n'y a qu'un seul élément dans PILE l'algorithme est terminé. Table de documentation 5.

8,3.3,3 - Lecture des têtes de modules logiques de traitement

Dans le cas d'option R ou C l'exécution du programme PDIC se termine par la lecture des cartes qui indiquent les têtes de module qui devront faire l'objet d'un nom de table à l'édition. L'adresse d'un paragraphe proposé est calculée et la variable TIPE qui lui est affectée évaluée. Le paragraphe est traité comme une entrée de circuit supplémentaire.

La colonne TIPE qui résume le code associé à chaque paragraphe du programme est recopiée dans le fichier PARAGRAPHE des noms des paragraphes COBOL : variable PREM-REF.

---

* Chaque séquence d'instructions groupées fait l'objet d'une étude permettant de déterminer les ruptures de séquences. Si une suite d'instructions se termine par un GO TO ou un STOP RUN le pointeur relatif à cette instruction est modifié :

8 devient 2 et 9 devient 3.

Les références à des instructions de ruptures de séquences seront ainsi reconnues sans qu'il soit nécessaire de lire l'instruction dans le fichier des souches. Une instruction EXIT est modifiée en une instruction EXIT nom du paragraphe. La référence à cette instruction dans le fichier PROG-REFE a été indiquée dans PROG-MASK avec la valeur de pointeur caractéristique 2. On ajoute une instruction :

NEXT SENTENCE

à une suite d'instructions sans rupture de séquence. Conventionnellement, la

référence d'une telle instruction est 8888. Elle sera utilisée lors du chaînage des tables atomiques d'un paragraphe.

La lecture dans PROG-MASK d'un terme de clôture de phrase 0 0000 provoque le passage à la phase de codage des tables atomiques. Nous résumerons le module de lecture d'un paragraphe par la table 1.

TABLES DE DOCUMENTATION

Table 1 LECTURE — Lecture des termes d'un paragraphe

TOUT LE PROGRAMME EST LU	Ø	N	N	N	N	N	N	N	N	N	N	N	N	N
LE POINTEUR DU TERME EST		0	0	0	0	0	1	1	1	2	9	9	9	Autre
C'EST LE PREMIER PARAGRAPHE							Ø	N	N					
C'EST LE DEBUT D'UNE PHRASE											Ø	N	N	
LE PARAGRAPHE QUI VIENT D'ETRE TRAITE APPELLE LE SUIVANT EN SEQUENCE								Ø	N					
DEBUT D'UNE PHRASE		Ø	N	N	N	N								
COURS DE PHRASE CONDITIONNELLE			Ø	Ø	N	N								
COURS DE PHRASE INCONDITIONNELLE					Ø	N						Ø	N	
C'EST LA PARENTHESE DE FIN DE PHRASE			Ø	N										
LIRE UN TERME DANS PROG-MASK		*	*	*	*	*	*	*	*	*	*	*	*	*
FERMER LA DERNIERE TABLE ETUDIEE	*							*	*					
EXECUTER LE REGROUPEMENT DES TABLES ATOMIQUES DU PARAGRAPHE	*							*	*					
OUVRIR UNE TABLE INCONDITIONNELLE											*			
FERMER UNE TABLE INCONDITIONNELLE					*									
RECOPIER LE TERME DANS T - TD				*									*	*
RECOPIER LE TERME DANS TS - TDS											*	*		
MENTION DE L'ARC DANS LE DICTIONNAIRE								*						
RAPPELER DANS BASE LA REFERENCE										*				
OUVRIR UNE TABLE CONDITIONNELLE RAPPELER DANS SUITE LA POSITION		*												
RAPPELER DANS PILE LA POSITION DES IF THEN ELSE ET COMPTER DANS IEL L'EQUILIBRE IF ... ELSE				*										
CODER LA PHRASE EN UNE TABLE ATOMIQUE			*											
INITIALISER LE TRAITEMENT D'UNE PHRASE		*	*											
INITIALISER LE TRAITEMENT D'UN PARAGRAPHE							*	*	*					
SIGNALER L'ERREUR						*								
GO TO LECTURE GO TO ENTREES DE CIRCUITS	* *	*	*	*	*	*	*	*	*	*	*	*	*	*

Table 2 CODAGE Codage d'une table atomique

Il y a autant de IF que de ELSE dans la phrase	Ø	N
CONSTRUIRE LA RUPTURE DE LA DERNIERE SEQUENCE LUE	*	*
COSTRUIRE IEL TERMES : 0 0003 2 8888		*
FERMER LA PHRASE : 0 0000	*	*
I DECRIT LES TERMES DE LA PHRASE SAUF LE PREMIER : EXECUTION DE COD2	*	*
FERMER LA TABLE RESULTANTE	*	*
EXIT CODAGE	*	*

Table 3 COD2 Construction des séquences

LE TERME I INDIQUE THEN	Ø	Ø	N	N	N	N	N	N	N	N	N	N	N
LA PILE MEMO EST VIDE	Ø	N											
LE TERME I INDIQUE UNE CONDITION			Ø	N	N	N	N	N	N	N	N	N	N
LE TERME I INDIQUE UNE BRANCHE													
OUI 5				Ø	Ø	Ø	Ø	Ø	N	N	N	N	N
NON 4									Ø	Ø	Ø	Ø	Ø
SOUS-TABLE COD22													
LE TERME I3 EST IF				Ø	Ø	N	N	N	Ø	Ø	N	N	N
LE TERME EST LE PREMIER DE LA SEQUENCE				Ø	N				Ø	N			
LE TERME INDIQUE ELSE OU LA FIN DE PHRASE						O	N	N			Ø	N	N
LE TERME INDIQUE THEN							Ø	N				Ø	N
FERMER LA TABLE	*												
METTRE DANS I LE SOMMET DE LA PILE MEMO		*											
A L'AIDE DU TABLEAU PILE RECHERCHER													
LE DEBUT I3 DE LA SEQUENCE OUI				*	*	*	*	*					
LE DEBUT I3 DE LA SEQUENCE NON									*	*	*	*	*
SOUS-TABLE COD22													
RAPPELER LA POSITION DU IF AU SOMMET DE MEMO				*	*				*	*			
METTRE I3 DANS I				*					*				
METTRE I3 + 1 DANS I. LE POINTEUR DE LA CONDITION DEVIENT 6					*					*			
RECOPIER LE TERME DANS TD TDS			*					*					
AJOUTER 1 A I3													
SIGNALER UNE ERREUR							*					*	
GO TO COD22								*					*
EXIT CODAGE	*												
EXIT COD2		*	*	*	*	*			*	*	*		
FIN							*					*	

Table 4 LECTURE-GRAPHE Construction du dictionnaire

IL RESTE UN ENREGISTREMENT A LIRE DANS GRAPHE	ø	ø	ø	ø	ø	ø	ø	ø	ø	N
LE CODE LU DANS L'ENREGISTREMENT EST	0	1	1	1	2	2	2	3	4	
LE TIPE ASSOCIE AU NOM EST		2	3	Aut	1	3	Aut			
LIRE UN ENREGISTREMENT DE GRAPHE	*	*	*	*	*	*	*	*	*	
CONSTRUIRE L'ADRESSE HC DU NOM	*	*	*	*	*	*	*	*	*	
RAPPELER L'ADRESSE DANS HC1	*									
LE TIPE DEVIENT		3	3	1	3	3	2	3		
MARQUER DANS LE DICTIONNAIRE A LA PREMIERE COLONNE LIBRE DE LA LIGNE HC1										
1000 + HC		*	*	*						
2000 + HC					*	*	*			
3000 + HC								*		
HC									*	
GO TO LECTURE (Table 1)										*
GO TO LECTURE-GRAPHE	*	*	*	*	*	*	*	*	*	

Table 5 — ENTREES-DE-CIRCUITS — Cheminement dans le graphe

SOUS-TABLE PARAGRAPHE-SUIVANT											
IL N'Y A PLUS QU'UN SEUL SUIVANT A ETUDIER	Ø	Ø	Ø	Ø	Ø	N	N	N	N	N	N
LE PARAGRAPHE HC EST UNE FIN DE PERFORM	Ø	N	N	N	N						
IL N'Y A PLUS DE SUIVANT						Ø	Ø	N	N	N	N
LA LIAISON INDIQUE UN PERFORM		Ø	Ø	N	N			Ø	Ø	N	N
LA PILE A UN SEUL ELEMENT						Ø	N				
LE PARAGRAPHE HC1 EST DEJA DANS PILE		Ø	N	Ø	N			Ø	N	Ø	N
METTRE L'ADRESSE DU PREMIER PARAGRAPHE EN HC ET PILE (1)	*	*	*	*	*	*	*	*	*	*	*
SOUS-TABLE PARAGRAPHE-SUIVANT											
CONSTRUIRE L'ADRESSE HC1 D'UN SUIVANT DE HC		*	*	*	*			*	*	*	*
RAPPELER DANS MEMO L'ADRESSE DU PARAGRAPHE FIN ET DANS SUITE LA POSITION DE L'APPEL		*	*					*	*		
SUPPRIMER LE DERNIER ELEMENT DE PILE	*						*				
INDIQUER DANS TIPE L'ENTREE DE CIRCUIT		*		*				*		*	
GO TO PARAGRAPHE-SUIVANT	*	*	*	*	*		*	*	*	*	*
FIN						*					

8.4 - REGROUPEMENT DES PARAGRAPHES, PROGRAMME PD2A

Pour regrouper les tables de décisions qui codent chaque paragraphe COBOL, ce programme utilise :

- Le fichier PROG-DEF du codage des tables.

- Le fichier PARAGRAPHE contenant les noms des paragraphes, par adressage dispersé, et leur type suivant le code :

1 pour un paragraphe qui doit être un titre de table. Il est adressable par une rupture de séquence GO TO ou par une action PERFORM de la souche actions

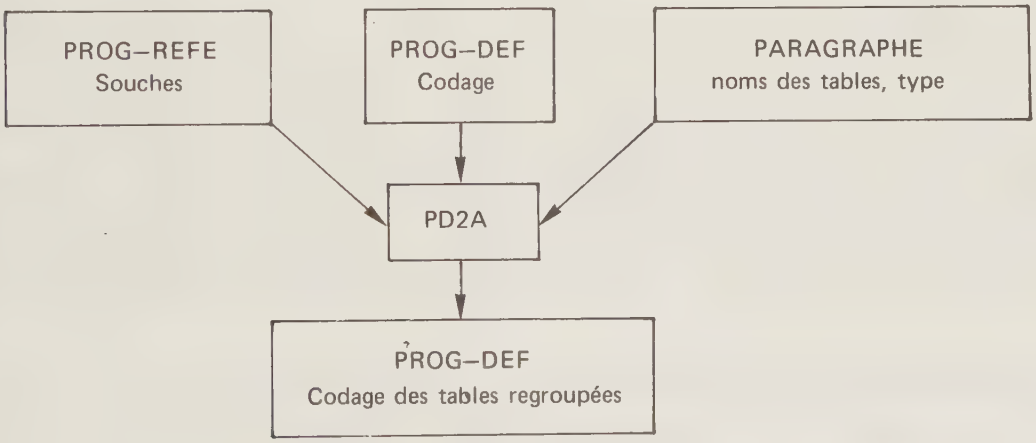
2 pour un paragraphe qui se trouve être un fin de module appelé par un verbe PERFORM

3 pour un paragraphe qui doit rester isolé

4 pour un paragraphe qui doit être réuni à toutes les tables qui l'appellent. Il s'agit d'un noeud de losange.

- Le fichier PROG-REFE contenant les textes des souches.

En résumé, le schéma suivant indique les transferts d'information entre les fichiers :



8.4.1 - Module I. Chargement des tables

Le codage des tables de tout le programme à étudier est chargé en mémoire centrale dans le tableau :

```
01 PROGRAMME-C.

    02 PG-T OCCURS 8000 PICTURE 9.

    02 PG-N COMP-1 OCCURS 8000 PICTURE 9(4).
```

La position de chaque tête de table est rappelée d'une part dans le tableau LIEU à l'adresse du nom du paragraphe, d'autre par en séquence dans le tableau SUITE. Ce double rangement permet de repérer rapidement :

- la position de la table d'un paragraphe dont on connaît le nom,

- la position des tables successives du programme.

Enfin au cours du chargement la longueur de chaque table est mémorisée dans le tableau LONG. Ce renseignement sera utilisé pour évaluer avant regroupement la longueur de la table résultante.

8.4.2 - Module II. Premières tables

Pour regrouper les tables on recherche en séquence dans le tableau SUITE, celles dont le type est 1 ou 3 ; le type est lu dans le fichier PARAGRAPHE. Les tables ainsi choisies sont celles sur lesquelles les regroupements sont entrepris. L'algorithme de regroupement est alors introduit.

Les termes de la table sont recopiés dans le fichier PROG-DEF, un par un. A la rencontre d'une instruction de rupture de séquence, le pointeur du terme vaut 2 ou 3, l'instruction de rupture est lue dans le fichier des souches. Le nom du paragraphe appelé est déterminé et son adresse dans PARAGRAPHE est calculée. Si le type de ce paragraphe est 1 ou 3 le regroupement n'a pas lieu et la lecture de la table se poursuit en séquence. Sinon le regroupement est entrepris après évaluation de la longueur de la table entière. Celle-ci doit être inférieure à un maximum fixé par la valeur du paramètre LMAX. Le regroupement des deux tables conduit aux mêmes opérations que dans la phase 1C précédente :

- empilement de l'indice de description de la table d'appel,

- détermination du premier pointeur de la table appelée.

La rencontre d'une parenthèse de fin de table 0000 entraîne la reprise de la table d'appel s'il y en a une, c'est-à-dire si la pile n'est pas vide.

8.4.3 - Module III. Tables noeud de losange

A la fin de la lecture du programme dirigée par le tableau SUITE, toutes les tables n'ont pas été utilisées. Nous le montrerons sur l'exemple suivant :

```
P1.

      PERFORM P2 THRU P3.

P2.

      ...   ...

                ...

P3.   EXIT.

P4.   STOP RUN.
```

Le module P2 P3 est atteint d'une part par l'intermédiaire d'un verbe PERFORM, d'autre part directement par passage de contrôle P1 - P2.

Le paragraphes	ont pour type :
P1	1
P2	1
P3	2
P4	4

Le paragraphe P4 est un noeud de losange du graphe de ce programme, cependant il ne fait l'objet d'aucune instruction de rupture qui le référence explicitement. La table de décisions de ce paragraphe n'a pas été étudiée au cours de la première lecture du programme. Une deuxième lecture permettra de traiter les tables ainsi atteintes.

D'autres tables correspondant à des noeuds de losange du graphe doivent encore être rappelées au cours de la deuxième lecture. Dans le cas où un regroupement n'est pas entrepris parce que la table résultante est trop longue, un noeud de losange fait l'objet d'une rupture de séquence. Il doit donc correspondre à une table adressable, éditée séparément.

Les tables qui doivent faire l'objet d'une étude en seconde lecture sont signalées par le type qui est associé à leur nom. Lors de la première lecture les types des tables sont modifiés comme suit :

1	devient	5
2		6
3		7
4		8

Si une table de type 4 ou 2 ne peut être regroupée pour une raison de taille de la table résultante le type est modifié en 9. Ainsi la seconde lecture conduit à rechercher les tables restantes de type 4 ou 9. L'algorithme de chaînage est entrepris comme dans la première lecture.

TABLES DE DOCUMENTATION

Table 1 MODULE I Chargement du codage des tables

IL RESTE DE LA PLACE POUR METTRE UN TERME EN MEMOIRE	Ø	Ø	Ø	Ø	N
TOUT LE PROGRAMME EST LU	Ø	N	N	N	
LE TERME EST CELUI DU NOM DE LA TABLE		Ø	Ø	N	
C'EST LA PREMIERE TABLE		Ø	N		
LIRE UN TERME DANS PROG-DEF		*	*	*	
LE METTRE EN MEMOIRE		*	*	*	
SIGNALER LE DEPASSEMENT					*
AJOUTER 1 A LA LONGUEUR DE LA TABLE		*	*	*	
METTRE EN MEMOIRE LA LONGUEUR DE LA TABLE PRECEDENTE TABLEAU LONG			*		
METTRE EN MEMOIRE LA POSITION DU TERME TABLEAUX SUITE ET LIEU		*	*		
INITIALISER LE COMPTEUR DE LONGUEUR DE TABLE		*	*		
GO TO MODULE I		*	*	*	
GO TO MODULE II	*				
FIN					*

Table 2 MODULE II Première lecture

tout le tableau suite a été décrit LE TYPE DE LA TABLE EST	Ø 1	Ø 3	Ø Autre	N
LE TYPE DEVIENT	5	7		
ECRIRE L'ENREGISTREMENT DE PARAGRAPHE	*	*		
EDITER LES 3 TERMES D'OUVERTURE DE LA TABLE	*	*		
SOIT I2 LA POSITION DU PREMIER TERME	*	*		
AJOUTER 1 A N2			*	
GO TO ECRITURE	*	*		
GO TO MODULE II			*	
GO TO MODULE III				*

Table 3 MODULE III Deuxième lecture

IL RESTE DES TABLES A LIRE DANS LIEU LE TYPE DE LA TABLE EST 4 OU 9	Ø Ø	Ø N	N	
SOIT I2 LA POSITION DE LA TABLE	*			
EDITER LES 3 TERMES D'OUVERTURE DE LA TABLE	*			
GO TO ECRITURE	*			
GO TO MODULE III		*		
FIN			*	

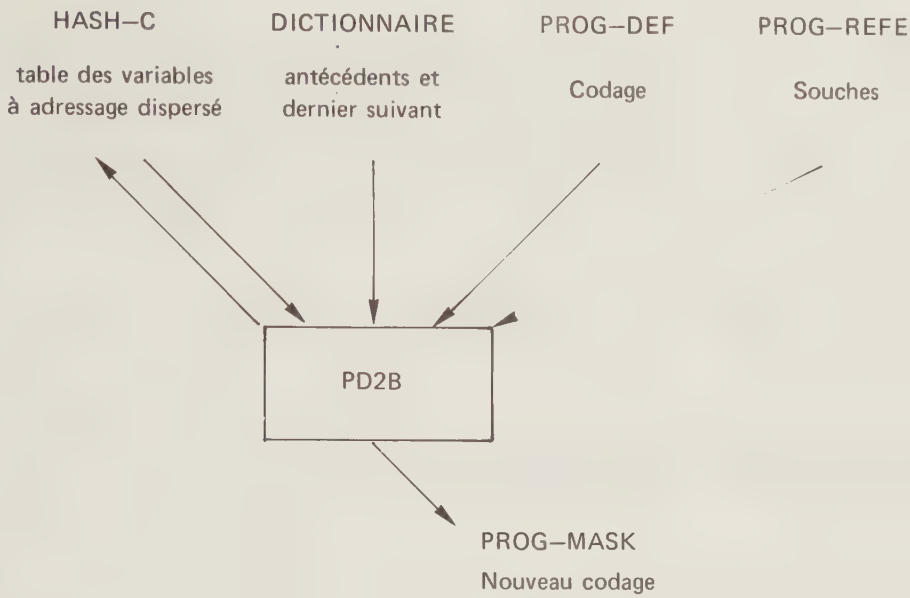
Table 4 ECRITURE Analyse des termes d'une table

LE TERME I2 INDIQUE UNE RUPTURE DE SEQUENCE	Ø	Ø	Ø	Ø	Ø	Ø	Ø	N	N	N	N
LE TERME I2 INDIQUE LA FIN D'UNE TABLE								Ø	Ø	Ø	N
LA RUPTURE EST EXIT OU STOP	Ø	N	N	N	N	N	N				
LA RUPTURE EST GO TO		Ø	Ø	Ø	N	N	N				
LA PILE EST VIDE								Ø	Ø	N	
C'EST LA PREMIERE LECTURE								Ø	N		
LE TYPE DE LA TABLE EST 1 3 5 7 9		Ø	N	N	Ø	N	N				
LA DIMENSION DE LA TABLE RESULTANTE EST INFERIEURE A LMAX			Ø	N		Ø	N				
SOIT I2 LE SOMMET DE LA PILE										*	
FERMER LA TABLE								*	*		
CONSTRUIRE LE NOM DE LA TABLE SON ADRESSE ET LA POSITION I2		*	*	*	*	*	*				
DETERMINER LE PREMIER POINTEUR			*			*					
LE TYPE DE LA TABLE EST 9				*			*				
LE TYPE DE LA TABLE EST 6 OU 8			*			*					
ECRIRE L'ENREGISTREMENT DE PARAGRAPHE			*	*		*	*				
EDITER LE TERME DANS PROG-DEF	*	*			*						*
AJOUTER 1 A I2	*	*			*						*
GO TO ECRITURE	*	*	*	*	*	*	*			*	*
GO TO MODULE II								*			
GO TO MODULE III									*		

8.5 - PROGRAMME PD2B. RUPTURES SEMANTIQUES

Le rôle de ce programme est de pointer les occurences de ruptures sémantiques dans le codage des tables tel qu'il est établi par la passe PD2A ou PD1C dans le fichier PROG-DEF. Le programme utilise le fichier des souches

d'actions et de conditions PROG-REFE, et les fichiers HASH-C et DICTIONNAIRE, construits à la lecture de la DATA DIVISION, pour traduire les hiérarchies entre noms de variables, créées par les arborescences de cette division. Finalement les transferts d'information entre les fichiers et le programme peuvent être schématisés comme suit :

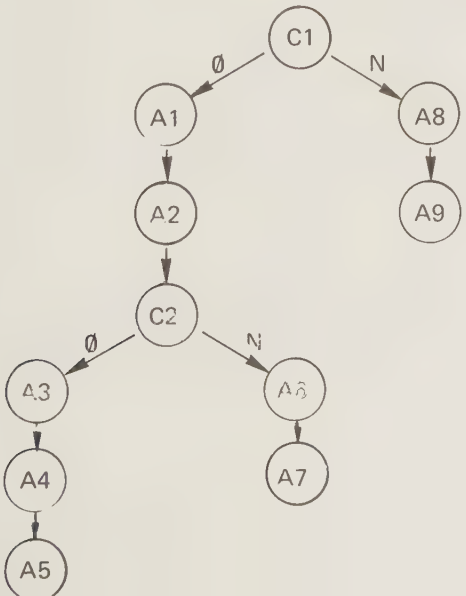


Le programme contient quatre modules fonctionnels distincts.

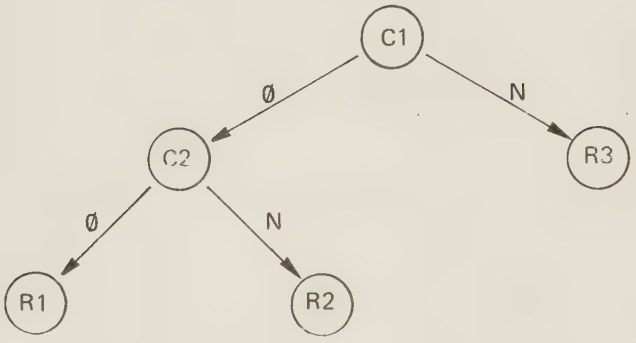
8.5.1 - Lecture et mise en mémoire d'une table

Le premier d'entre eux lit le codage d'une table dans le fichier d'entrée. Il charge en mémoire centrale cette table sous la forme d'un codage d'arbre binaire. Une série de pointeurs fait la liaison entre un sommet terminal de cette arborescence générique et la suite des actions qu'il représente.

Ainsi l'arborescence suivante :



est mise en mémoire sous la forme :



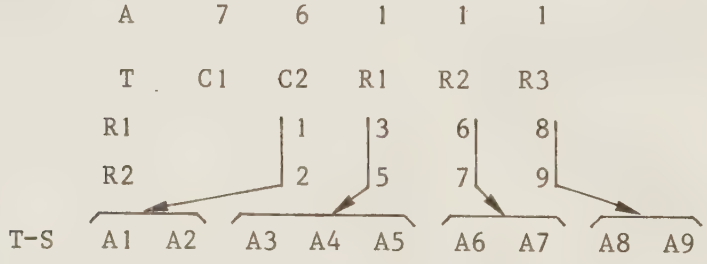
et le jeu de pointeurs mentionné fait correspondre :

à	les actions :
R1	3 , 4 , 5
R2	6 , 7
R3	8 , 9

et par analogie fait correspondre à la condition 2 les actions 1 et 2 qui la précèdent. Le codage de l'arborescence citée pourrait être :

dans PROG-DEF : 7 C1 8 A1 9 A2 6 C2 8 A3 9 A4 9 A5 8 A6 9 A7 8 A8
9 A9 0 00

et en mémoire centrale la forme sera :



Les noms A T R1 R2 T-S sont ceux des tableaux du programme. On remarque que la ligne T contient le codage préfixé de l'arborescence binaire générique de la table. Le programme réalise cette mise en mémoire au moyen d'un module chargeur.

8.5.2 - Recherche des variables réceptrices

Une rupture sémantique ne peut se trouver que dans le cas où l'évaluation d'une condition intervient après des actions (présence d'un 6 dans le codage). En l'absence d'une telle circonstance il ne peut y avoir de rupture et la table est directement réécrite dans le fichier de sortie PROG-MASK. Il faut ensuite que les variables dont la valeur est affectée par les actions, nous les avons appelées variables réceptrices, interviennent dans l'évaluation de la condition.

Le module que nous présentons ici recherche les variables réceptrices dans le texte des actions exécutées avant une condition.

De telles actions sont pointées dans les lignes R1 R2 sous une condition c'est-à-dire sous un terme 6 dans la ligne A. L'action est lue dans le fichier PROG-REFE et suivant le verbe COBOL la recherche d'une variable réceptrice est entreprise.

On calcule alors l'adresse de cette variable par un hash-code. L'utilisation des tableaux d'antécédents et de suivants du fichier DICTIONNAIRE permet de déterminer les noms de toutes les variables simples ou de groupe dont la valeur est modifiée. Ce sont généralement des variables réceptrices. La mise en mémoire des variables réceptrices se fait à l'aide d'une clé affectée à chaque nom de variable (Tableau CLE). A la fin de l'étude de toutes les actions précédent le 6, les variables réceptrices sont telles que la clé qui leur est affectée vaut 2. Pour les autres variables du programme la clé vaut 1.

8.5.3 - Recherche des variables intervenant dans l'évaluation d'une condition

Ce module est exécuté pour toutes les conditions descendant de la condition sous 6, signalée au module précédent, celle-ci comprise. La ligne T rappelle les numéros de clé d'accès dans le fichier des souches. Une condition est lue. Les mots intervenant dans le texte sont séparés, et les variables distinguées des mots COBOL réservés et des littéraux. Une variable isolée, son adresse de hash-code est évaluée. La clé affectée à la variable est alors consultée. S'il apparaît qu'il s'agit d'une variable réceptrice une rupture sémantique est trouvée. Celle-ci est indiquée sur la ligne A. Le pointeur de la condition est modifié :

6 devient 4 et 7 devient 5

selon la syntaxe de codage détaillée au chapitre 4.

8.5.4 - Ecriture d'une table

En fin d'étude la table est éditée dans le fichier prog-mask. La forme retenue est analogue à la forme de la table à la lecture. La transposition des tableaux A , T , R1 , R2 , T-S en un codage syntaxiquement correct est réalisée de façon simple. Le codage obtenu fait apparaître l'arite des conditions (degré extérieur du sommet) et le numéro d'entrées des actions. Il est ainsi compatible avec le codage des tables à entrées étendues. Les programmes de la phase 3 suivante ne lisent que ce type de codage.

TABLES DE DOCUMENTATION

Table 1 MODULE I Chargement d'une table

LE PROGRAMME EST ENTIEREMENT LU	Ø	N	N	N	N	N	N	N	N
LE TERME INDIQUE :									
LE NOM D'UNE TABLE		Ø	N	N	N	N	N	N	N
LA FIN D'UNE TABLE			Ø	Ø	N	N	N	N	N
LE POINTEUR VAUT					2/8	3/9	6	7	Aut
IL N'Y A PAS EU DE CONDITIONS			Ø	N					
LIRE UN TERME DANS PROG-DEF		*	*	*	*	*	*	*	*
EDITER LES TROIS PREMIERS TERMES DE LA TABLE		*							
EDITER LA TABLE SANS MODIFICATIONS			*						
AJOUTER UN TERME DANS LES LIGNES A T					*			*	
AJOUTER 1 AU NOMBRE DE REGLES NR. METTRE NR DANS LA LIGNE T ET 1 DANS A					*				
METTRE LA REFERENCE DANS LA LIGNE T-S EN RAPPELER DANS R2 LA POSITION					*	*			
RAPPELER DANS R1 LA POSITION					*				
METTRE LA REFERENCE DANS LA LIGNE T							*	*	
METTRE LE POINTEUR DANS LA LIGNE A							*	*	
SIGNALER L'ERREUR									*
GO TO MODULE I		*	*		*	*	*	*	
GO TO MODULE II				*					
FIN	*								*

Table 2 — MODULE II Recherche des variables receptrices

LE TERME I DE LA LIGNE A EST 6	Ø	Ø	Ø	Ø	Ø	Ø	N
SOUS-TABLE RECH-VAR							
I3 EST SUPERIEUR A R2 (I)	Ø	N	N	N	N	N	
LE VERBE DE L'INSTRUCTION REFERENCEE EN T-S (I3) EST :							
MOVE		Ø	N	N	N	N	
PERFORM			Ø	N	N	N	
COMPUTE OU READ				Ø	N	N	
SET OU ACCEPT					Ø	N	
INITIALISER LES CLES A 1	*	*	*	*	*	*	
METTRE DANS I3 LA POSITION R1 (I) DE LA PREMIERE REFERENCE A UNE INSTRUCTION EXECUTEE AVANT LA CONDITION	*	*	*	*	*	*	
SOUS-TABLE RECH-VAR							
METTRE TOUTES LES VARIABLES SOUS-CLE			*				
CONSTRUIRE DANS CHAQUE CAS LE NOM DE LA VARIABLE RECEPTRICE		*		*	*		
CALCULER SON ADRESSE		*		*	*		
METTRE LA VARIABLE SOUS-CLE		*		*	*		
DE PROCHE EN PROCHE METTRE SES ANTECEDENTS SOUS CLE		*		*	*		
METTRE TOUS SES SUIVANTS SOUS CLE		*		*	*		
AJOUTER 1 A I3		*		*	*	*	
AJOUTER 1 A I							*
GO TO MODULE II							*
GO TO MODULE III	*		*				
GO TO RECH-VAR		*		*	*	*	

Table 3 MODULE III Recherche des ruptures sémantiques

LE TERME I2 EST UN DESCENDANT DE LA CONDITION I	Ø	Ø	Ø	Ø	Ø	Ø	N
LE TERME I2 DE LA LIGNE A INDIQUE UNE CONDITION	Ø	Ø	Ø	Ø	Ø	N	
SOUS-TABLE RECH-MOT							
TOUS LES MOTS DU TEXTE ONT ETE ETUDIES	Ø	N	N	N	N		
LE MOT EST UN MOT RESERVE COBOL		Ø	N	N	N		
LE MOT EST UN LITTERAL NUMERIQUE			Ø	N	N		
LA VARIABLE EST SOUS CLE				Ø	N		
LIRE LA CONDITION REFERENCEE EN T (I2)	*	*	*	*	*		
SOUS-TABLE RECH-MOT							
ISOLER UN MOT DANS LE TEXTE SUPPRIMER LES ( )		*	*	*	*		
CALCULER L'ADRESSE DE LA VARIABLE				*	*		
UNE RUPTURE SEMANTIQUE EST TROUVEE EN I2				*			
AJOUTER 1 A I2	*					*	
AJOUTER 1 A I				*			*
GO TO MODULE II				*			*
GO TO MODULE III	*					*	
GO TO RECH-MOT		*	*		*		

8.6 - PHASE 3

Les quatre programmes de la troisième phase forment un ensemble indépendant. Ils admettent un codage des tables de décisions qui obéissent à la grammaire définie pour représenter les tables à entrées étendues. Le premier programme, noté PD3A, charge les fichiers permanents qui résument toute l'information qui permet d'éditer les tables. Les textes de souche y sont inscrits dans l'ordre qui sera le leur à l'édition, et les phrases qui codent les tables font référence à ces textes ordonnés. Le second programme édite sur imprimante les tables telles qu'elles sont codées dans les fichiers permanents.

Les deux derniers programmes PD3C et PD3D traitent les directives d'édition données par l'utilisateur. A partir des mêmes fichiers permanents, ils créent un nouveau cycle du codage et des souches ; mis en mémoire dans les fichiers temporaires, ce nouveau cycle est ordonné par une nouvelle exécution de PD3A.

Un schéma général de fonctionnement des programmes de cette phase est figuré à la page suivante. Des schémas détaillés qui ont référence aux noms des fichiers sont fournis dans la description de chacun des quatre programmes.

8.6.1 - Programme PD3A

Ce programme étudie successivement deux des composants de la forme éditée d'une table : souche conditions et souche actions. Cette étude conduit à la génération d'une nouvelle forme codée. Nous distinguerons les 4 modules :

1) Chargement d'une phrase.
   - o Lecture.
   - o Derniers suivants. Ordre des sous-tables.

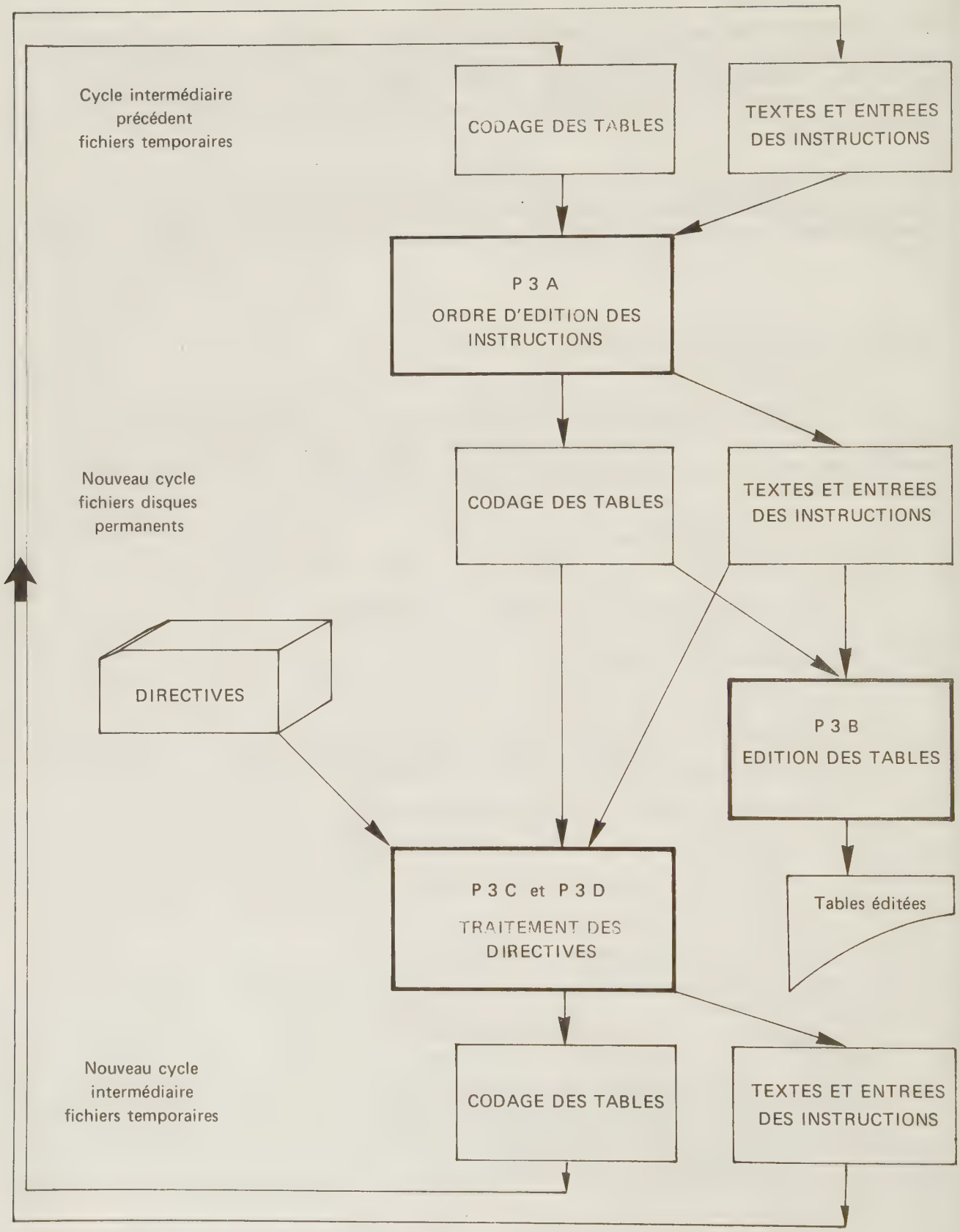
2) Souche conditions.
   - o Ordre des conditions.
   - o Recherche de branches inaccessibles.
   - o Construction des nouvelles références conditions.

3) Souche actions et ruptures de séquences.
   - o Construction de la matrice de connexion P.
   - o Calcul des puissances successives de P soit Q.
   - o Actions identiques.
   - o Ordre d'édition des actions.
   - o Ruptures de séquences.

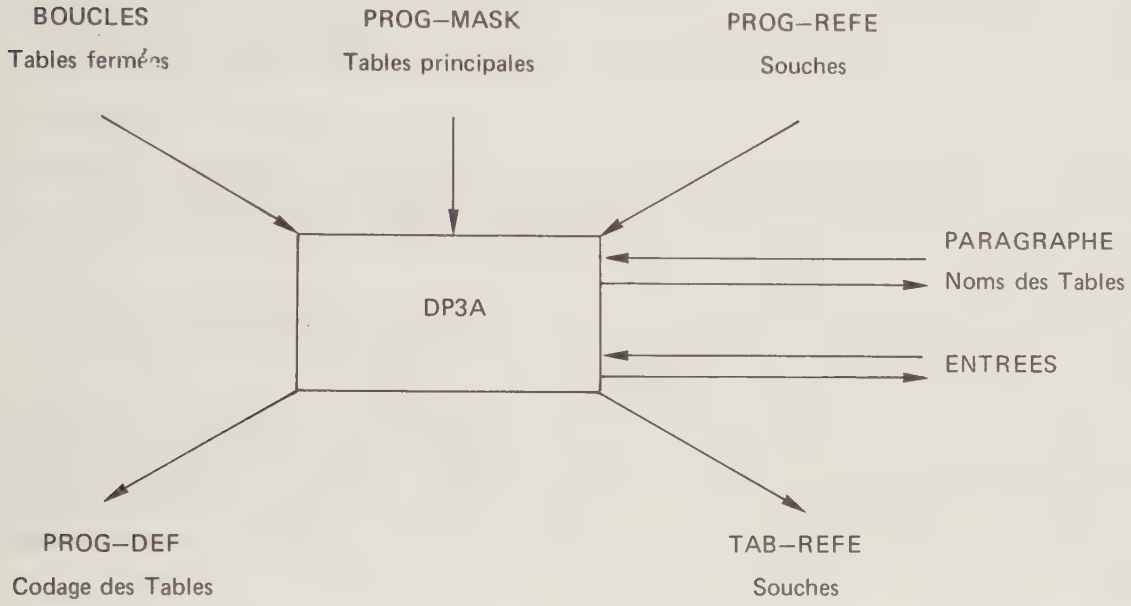
4) Edition de la nouvelle phrase de codage.

La première exécution du programme PD3A (tables à entrées limitées) conduit à étudier les tables principales contenues dans le fichier PROG-MASK et les tables fermées générées par PD1B à l'occasion des verbes PERFORM ... UNTIL ... Ces dernières sont contenues dans le fichier BOUCLES et n'ont fait l'objet d'aucune transformation. Le résultat de l'exécution de PD3A est résumé

Figure 8-6 – ORGANIGRAMME DE LA PHASE 3



dans les fichiers permanents. Pour permettre les exécutions suivantes, le fichier BOUCLES est entièrement vidé après étude. Le schéma général de fonctionnement du programme est le suivant :



8.6.1.1 - Chargement d'une phrase

Le chargement d'une phrase en mémoire centrale, s'accompagne de la construction des références minimum et maximum aux textes de souches. Celles-ci seront utilisées pour la mise en fichier du nouvel ordre des textes. Chaque terme du codage est lu et mis en mémoire dans le tableau formé des variables :

```
03 T PICTURE 9.

03 AR PICTURE 9(2).

03 A COMP-1 PICTURE 9(4).
```

1 000 termes peuvent être enregistrés. Au cours du chargement le nombre NR de règles de la table est évalué. Rappelons que à chaque rupture de séquence correspond une règle différente. Le numéro de la règle associée à une rupture de séquence est rappelé dans la ligne S.

8.6.1.1.2 - Derniers suivants. Ordre des sous-tables

Cette ligne S contient la position du dernier suivant associé à chaque terme qui n'est pas une rupture de séquence. Le chargement d'une table se termine par la construction de l'ordre d'édition des sous-tables titrées ou non qui doivent être distinguées sur la forme éditée. La ligne ST fait correspondre à chaque terme du codage le numéro d'ordre de la sous-tables à laquelle il appartient.

L'algorithme qui permet de construire la ligne des derniers suivants a été décrit au chapitre 4 et modifié au chapitre suivant de façon à tenir compte du nombre quelconque des suivants d'une condition : table P4.

L'ordre d'édition des sous-tables est évalué en recherchant les imbrications de sous-tables. Le codage est décrit de gauche à droite, chacun des termes pointé par 5 ou 4 conduit à l'édition d'une sous-table non titrée l'ordre d'imbrication est augmenté de 1. Il est valable pour tous les termes jusqu'au dernier suivant du titre. Un terme pointé 1 correspondant au nom d'une sous-table titrée augmente également l'ordre d'imbrication jusqu'à son dernier suivant. Une précaution est prise de façon à ce que une sous-table non titrée ne puisse avoir le même ordre d'édition qu'une sous-table titrée. Le chargement de la ligne ST est décrit par la table P5.

TABLES DE DOCUMENTATION

Table 1

LECTURE										
J VAUT	1	1	2	2	3	3	3	3	3	3
SOUS-TABLE 1										
LA REFERENCE EST NULLE	Ø	N	Ø	N	Ø	Ø	N	N	N	N
LE POINTEUR VAUT 1							Ø	N	N	N
LA REFERENCE INDIQUE UN APPEL DE LA TABLE SUIVANTE								Ø	N	N
LE POINTEUR INDIQUE UNE RUPTURE DE SEQUENCE									Ø	N
SOUS-TABLE 2										
LE DERNIER SUIVANT DU TITRE DE LA TABLE EST LE DERNIER TERME DE LA TABLE					Ø	N				
LIRE UN TERME DE LA PHRASE	*	*	*	*	*	*	*	*	*	*
SOUS-TABLE 1										
CONSTRUIRE LE NOM DE LA TABLE		*								
J		2	3							
SIGNALER UNE ERREUR DANS LE CODAGE				*						
CHARGER LA LIGNE S. I DE NT A ZERO					*	*				
METTRE LE TERME EN MEMOIRE							*	*	*	*
AJOUTER 1 AU NOMBRE DE REGLE								*	*	
MEMORISER L'APPEL EN SEQUENCE								*		
CONSTRUIRE LES REFERENCES MIN ET MAX									*	*
SOUS-TABLE 2										
SIGNALER UNE ERREUR DANS LE CODAGE						*				
CHARGER LA LIGNE ST					*					
GO TO LECTURE	*	*	*				*	*	*	*
FIN				*	*	*				

Table 2 — P4 — Chargement de la ligne S

LE POINTEUR DU TERME I INDIQUE : 1 une rupture de séquence 2 une action ou un nom de sous-table 3 une condition à n suivants	1	2	3	
EMPILER LA POSITION DU TERME	*			
FAIRE n-1 DEPILEMENTS			*	
LE DERNIER SUIVANT EST LE SOMMET DE LA PILE	*	*	*	
EXIT P4	*	*	*	

Table 3 — P5 — Ordre d'édition des sous-tables

LE POINTEUR DU TERME I EST 1,4 OU 5	Ø	Ø	Ø	Ø	Ø	Ø	N
SOUS-TABLE P50							
LE NUMERO COURANT (DEG + 1) A DEJA ETE RENCONTRE : I2 LE TITRE DE LA SOUS-TABLE	Ø	Ø	Ø	Ø	Ø	N	
LA SOUS-TABLE I EST TITREE	Ø	Ø	Ø	N	N		
LA SOUS-TABLE I2 EST TITREE	Ø	Ø	N	Ø	N		
LE TITRE EST LE MEME	Ø	N					
METTRE ST (I) EN DEG	*	*	*	*	*	*	
SOUS-TABLE P50							
AJOUTER 1 A DEG	*	*	*	*	*	*	
CALCULER LE NUMERO D'ORDRE MAXIMUM	*				*	*	
EMPILER DEG ET I	*				*	*	
MARQUER TOUS LES SUIVANTS DE I AVEC DEG	*				*	*	
EXIT P5	*				*	*	*
GO TO P50		*	*	*			

8.6.1.2 - Souche conditions

La souche condition est étudiée de façon indépendante pour chacune des sous-tables qui devront être distinguées à l'édition.

8.6.1.2.1 - Ordre des conditions

Un terme de la phrase qui fait référence à une condition et pour lequel la ligne ST indique qu'il doit être édité dans la sous-table en cours d'étude est retenu. Le numéro de référence de la condition est comparé à ceux déjà retenus et mémorisés dans une liste. Si la référence n'y est pas trouvée, elle est ajoutée à la liste. Les textes de souche des conditions retenues sont comparés deux à deux ; si deux textes identiques sont trouvés, les entrées associées sont comparées, si elles sont encore identiques, les références à la deuxième condition sont supprimées dans le codage de la sous-table pour être remplacées par la première. La liste des conditions retenues est réduite. La vérification particulière décrite au paragraphe suivant est entreprise dans ce cas.

Au terme de l'étude des conditions de toutes les sous-tables, la liste contient l'ensemble des références aux conditions différentes de la table dans l'ordre qui sera adopté à l'édition.

8.6.1.2.2 - Branches inacessibles

Un cas de branche inaccessible a été présenté au paragraphe *4.3.3.1.2.* Une condition ne peut appartenir à une sous-table dont le titre lui soit identique. La vérification qu'un tel phénomène n'est pas présent dans le codage est rapide et entreprise systématiquement lorsque deux textes de conditions identiques sont trouvés. Dans le cas contraire, il faut chercher le numéro de la branche qui lie les deux conditions, dans l'ordre des suivants de la première. La branche correspondante à ce numéro d'ordre et descendant de la seconde condition est isolée. C'est la seule branche accessible. Elle doit être conservée. Toutes les autres branches descendant de la seconde condition sont supprimées. Le rattachement de la branche accessible entraîne une modification du pointeur de son titre. Dans le tableau suivant, la première ligne présente les pointeurs possibles de la deuxième condition, la première colonne, les pointeurs du titre de la branche accessible. Le nouveau pointeur de ce terme est déterminé par la case d'intersection.

	4	5	6	7
2	3	2	3	2
5	4	5	4	5
7	6	7	6	7
8	9	8	9	8

Le traitement ainsi subi par la phrase rend inutilisables les informations contenues dans les lignes S et ST puisque des parties du codage ont été déplacées. Ces lignes doivent être rechargées. La construction de l'ordre des conditions est reprise entièrement.

8.6.1.2.3 - Construction des nouvelles références

L'ordre d'édition des conditions construit pour toute la table, les références à ces conditions dans la phrase sont modifiées. Les textes de souches correspondant sont recopiées dans l'ordre et en séquence dans le fichier permanent TAB-REFE. Une ligne notée RZ permet de rappeler que le changement des clés d'accès a été effectué sur le terme correspondant de la phrase.

TABLES DE DOCUMENTATION

Table 4 SEC2	Ordre des conditions				
LE I° TERME EST UNE CONDITION DE LA SOUS-TABLE	Ø	Ø	Ø	Ø	N
LA REFERENCE EST DEJA DANS LA LISTE	Ø	N	N		
LE TEXTE ET LES ENTREES DE LA CONDITION SONT IDENTIQUES A CEUX D'UNE CONDITION DE LA LISTE		Ø	N		
REMPLACER PAR CELLE DE LA LISTE LES REFERENCES A LA CONDITION DANS LA PHRASE		*			
TRAITEMENT DES BRANCHES INACCESSIBLES		*			
CHARGER LES LIGNES S ET ST		*			
AJOUTER LA REFERENCE DU TERME A LA LISTE			*		
EXIT SEC2	*		*	*	
GO TO SEC		*			

Table 5 SEC3 Branches inaccessibles

LE I° TERME EST UNE CONDITION	Ø	Ø	N
EXISTE-T-IL DANS LA SOUS-TABLE DE TITRE I UNE DEUXIEME OCCURENCE I3 DE LA CONDITION	Ø	N	
METTRE EN MEMOIRE LA POSITION DES TETES DE BRANCHES DESCENDANT DES 2 CONDITIONS	*		
EVALUER LE NUMERO DE LA BRANCHE QUI LIE LES DEUX CONDITIONS	*		
EVALUER LE NOUVEAU POINTEUR DU TITRE DE LA BRANCHE ACCESSIBLE	*		
RECOPIER LA BRANCHE ACCESSIBLE	*		
DECALER LE RESTE DE LA PHRASE DE CODE	*		
EXIT SEC3	*	*	*

8.6.1.3 - Souche action

L'étude de la relation d'ordre partiel qui permet de construire la souche action attachée à une sous-table distinguée à l'édition a été présentée au chapitre 3.

8.6.1.3.1 - Construction de la matrice de connexion P

Cette matrice binaire est initialisée à zéro. C'est la seule table à deux dimensions qui intervienne dans toute l'étude. Un vecteur LISTE associe à chaque ligne ou colonne une référence d'action. Cette liste est construite au cours de la lecture de la phrase qui code la sous-table en recherchant les numéros de référence différents. Deux actions A1 et A2 en séquence sont signalées par un 1 dans P à l'intersèction de la ligne associée à A1 et de la colonne associée à A2. Dans le codage, une séquence d'actions se termine par un terme pointé :

3 rupture de séquence

4 sous-table imbriquée distinguée à l'édition

1 sous-table titrée imbriquée.

La dernière action précédent une condition pointée 6 doit être liée à chacune des premières actions de chaque branche descendant de la condition. Ceci est réalisé à l'aide d'une pile qui rappelle la position dans la liste de l'action précédente et le dernier suivant de la condition. L'emploi d'une pile est imposé par la récursivité possible de séquence conditionnnelle.

8.6.1.3.2 - Puissances successives de P

Le calcul des puissances logiques successives de P est organisé de façon à ne pas imposer l'emploi d'une deuxième zone matricielle. Le résultat de l'évaluation d'un terme de la matrice produit est inscrit sur la matrice elle-même. Une matrice intermédiaire n'a donc plus de signification en termes de longueur des chemins dans le graphe G_a. A chaque itération le nombre IEQ d'éléments de la matrice dont la valeur est modifiée est compté. Le calcul est arrêté dès que IEQ est nul. Le résultat est une matrice qui rend compte de toutes les relations d'antécédence entre les actions de la liste construite précédemment.

A chaque itération, la matrice est étudiée élément par élément, soit :

P (I, I1). Si P (I, I1) vaut 1 sa valeur n'est pas modifiée. Sinon, s'il existe I2 tel que :

$$P(I2, I1) = P(I, I2) = 1$$

l'élément (I, I1) est mis à 1 et IEQ incrémenté.

8.6.1.3.3 - Actions identiques

Ce module a pour objet de supprimer les actions dont le texte est identique en simplifiant la souche actions. Rappelons le résultat qui permet cette simplification. Deux sommets dans G_a^* qui ne sont liés par aucun chemin peuvent être supprimés si les étiquettes qu'ils portent sont identiques. Un sommet nouveau est créé dans G_a^* . Il porte l'étiquette commune et appartient à tous les chemins auxquels appartenaient les deux sommets supprimés.

Les instructions sont lues dans l'ordre des références de la liste. La lecture d'un texte référencé dans la suite de la liste n'est entreprise que s'il n'existe aucune relation d'antécédence entre les deux actions qui vont être comparées. Dans le cas de textes identiques, les entrées des deux actions sont à leur tour comparées deux à deux. Un enregistrement d'entrées est construit, il résume les entrées des deux actions. Les arcs correspondants à chacune des entrées sont recherchés dans le codage et la référence à l'étiquette de chacun est modifiée dans la ligne AR. Enfin la matrice de connexion du graphe simplifié est déduite de P. Les 1 des lignes et colonnes correspondantes au deuxième sommet sont recopiés à leur place respective dans les ligne et colonne associées au premier sommet action.

Elles sont ensuite supprimées de la matrice. Un produit logique supplémentaire de P est réalisé. Le traitement se termine par la recherche dans la phrase de code des références à la deuxième action. Elles sont remplacées par la clé d'accès au texte commun.

8.6.1.3.4 - Ordre des actions à l'édition

Les textes qui seront édités sont maintenant déterminés. Les relations d'antécédence entre eux sont résumées par la matrice de connexion P. Un vecteur appelé LISTE permet d'associer un texte à une ligne et une colonne de P. Le module SAC4 a pour objet de construire l'ordre d'édition des actions de la sous-table en cours d'étude.

La matrice P est décrite ligne par ligne. Si une ligne de P ne contient que des éléments nuls, l'action correspondante n'admet aucun antécédent dans la sous-table et peut être imprimée en premier. Nous avons signalé que le graphe G_a n'est pas connexe ; plusieurs lignes de P peuvent donc être entièrement nulles. Les actions correspondantes seront éditées dans l'ordre qu'elles occupent dans LISTE. Les lignes et colonnes de P peuvent être effacées, elles sont en pratique annulées par un vecteur de clés. La recherche des actions suivantes, c'est-à-dire qui n'ont qu'un antécédent est entreprise par étude des lignes de P et des éléments de chaque ligne dont la colonne n'est pas annulée.

L'étude de la souche actions d'une sous-table d'édition se termine par l'édition, dans le fichier de sortie TAB-REFE, des textes retenus, dans l'ordre qu'ils occuperont à l'édition de la table entière. Les clés d'accès sont modifiées dans la phrase de codage de façon à faire référence aux textes créés. Module SAC5.

8.6.1.3.5 - Ruptures de séquences

Le traitement des ruptures de séquences de la table entière se résume à la recherche des références différentes trouvées dans la phrase de codage. Les textes sont comparés deux à deux ; aucune relation d'ordre n'intervenant entre deux ruptures de séquences la matrice P n'est pas construite. Les ruptures de séquence sont éditées dans TAB-REFE dans l'ordre qu'elles occupent dans la liste, et les références dans la phrase modifiées en conséquence.

8.6.1.4 - Edition de la nouvelle phrase de codage

Le traitement d'une table se termine par l'édition dans le fichier permanent PROG-DEF de la phrase qui code la table. Ce codage fait référence

aux textes de souche contenus dans le fichier TAB-REFE.

TABLES DE DOCUMENTATION

Table 6 SAC1 Matrice de connexion

LE TERME I EST UNE ACTION DE LA SOUS-TABLE	Ø	Ø	Ø	Ø	Ø	N
LA REFERENCE DU TERME I EST DANS LA LISTE	Ø	Ø	N	N	N	
IL Y A DEJA 63 ACTIONS DANS LA SOUS-TABLE			Ø	N	N	
LA SEQUENCE EN COURS EST CONDITIONNELLE	Ø	N		Ø	N	
SOIT I1 LA COLONNE DE P ASSOCIEE A LA REFERENCE DU TERME I	*	*				
AJOUTER LA REFERENCE I A LA LISTE AUGMENTER LA DIMENSION DE P				*	*	
LIER L'ACTION A LA PRECEDENTE DU 6, POINTEE PAR LE SOMMET DE LA PILE	*			*		
SIGNALER LE DEPASSEMENT DE CAPACITE			*			
EXIT SAC1						*
FIN			*			
VERS TABLE SUIVANTE	*	*		*	*	

Table 7 SAC11 Suivant de l'action

LE TERME I2 SUIVANT I DANS LE CODAGE EST UNE ACTION	Ø	Ø	Ø	N	N	
LE TERME I2 EST UNE CONDITION POINTEE 6				Ø	N	
LA REFERENCE DU TERME I2 EST DANS LA LISTE	Ø	N	N			
IL Y A DEJA 63 ACTIONS DANS LA LISTE		Ø	N			
EMPILER SUCCESSIVEMENT L'INDICE I1 ET LE DERNIER SUIVANT DE LA CONDITION S(I2)				*		
SOIT I3 LA LIGNE DEP ASSOCIEE A LA REFERENCE DU TERME I2	*					
AJOUTER LA REFERENCE I2 A LA LISTE ET AUGMENTER LA DIMENSION DE P			*			
MARQUER 1 DANS P (I3, I1)	*		*			
SIGNALER LE DEPASSEMENT DE CAPACITE		*				
EXIT SAC1	*		*	*	*	
FIN		*				

Table 8 SAC3 Textes actions identiques

SOUS-TABLE RETOUR I1 = I	Ø	N
LIRE LES ENTREES ET LE TEXTE DE LA Ième ACTION DE LA LISTE	*	*
INDICE DE LA FIN DE LISTE DANS I1	*	*
SOUS-TABLE RETOUR		
SOUSTRAIRE 1 A I1		*
EXECUTER SAC30		*
GO TO RETOUR		*
EXIT SAC3	*	

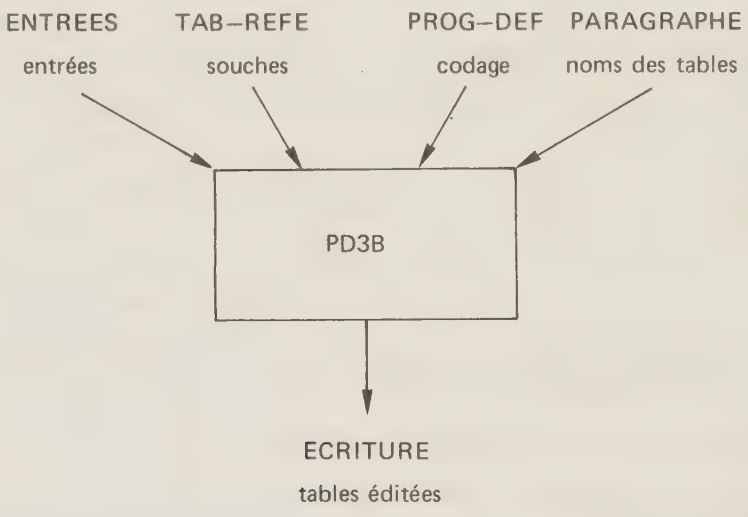
Table 9 SAC30 Simplification du graphe

LES ACTIONS I ET I1 DE LA LISTE SONT LIEES DANS LA MATRICE DE CONNEXION	Ø	N	N	N
LES TEXTES DES DEUX ACTIONS SONT IDENTIQUES		Ø	Ø	N
LES ENTREES DES DEUX ACTIONS SONT IDENTIQUES ET DANS LE MEME ORDRE		Ø	N	
CONSTRUIRE UN ENREGISTREMENT QUI RESUME LES ENTREES DES DEUX ACTIONS		*		
MODIFIER LES REFERENCES AUX ENTREES DANS LE CODAGE		*		
RECOPIER LES 1 DES LIGNE ET COLONNE I1 DE P SUR LES LIGNE ET COLONNE I		*	*	
SUPPRIMER LIGNE ET COLONNE I1		*	*	
DIMINUER DE 1 LA DIMENSION DE P		*	*	
EXECUTER UN PRODUIT LOGIQUE		*	*	
MODIFIER LES REFERENCES A L'ACTION I1 DANS LE CODAGE		*	*	
EXIT SAC30	*	*	*	*

8.6.2 - Programme PD3B, Edition des tables

8.6.2.1 - Présentation

Le programme PD3B réalise l'édition et la mise en page des tables de décisions telles qu'elles sont fournies par l'exécution de PD3A. Le programme ne crée aucun nouveau cycle de codage des tables. Un schéma de fonctionnement peut être dessiné :



Les fichiers d'entrées sont les fichiers permanents affectés à la mémoire de masse. Ces mêmes fichiers seront utilisés par PD3C.

Le traitement d'une table qui aboutit à son édition est organisé par les deux modules suivants :

8.6.2.2 - Chargement du codage en mémoire centrale et construction des lignes S qui pointe le dernier suivant de chaque terme, et ST qui indique le numéro d'ordre à l'édition de la sous-table à laquelle appartient chaque terme. Ce module est absolument identique au chargeur construit et décrit dans le programme PD3A.

8.6.2.3 - Elaboration des entrées d'une ligne et édition ligne par ligne de la table de décisions à entrées mixtes. Les tables de décisions qui suivent intéressent ce module essentiel. Enfin signalons la présence de nombreux auxiliaires de mise en page ou de construction. Nous citerons sans les décrire :

- Détermination de la longueur des textes de souche.
- Construction d'un interligne de la table.
- Edition du titre de la table.
- Lecture et copie sur la ligne d'un texte de souche.
- Edition des titres et séparation des sous-tables.
- Recherche de l'entrée numéro n attachée à une instruction donnée.
- Ouverture et fermeture d'une table. Reprise de l'édition d'une table sur plusieurs pages.

Les lignes sont éditées en ordre croissant de références à partir de la plus petite notée NO-MIN. Pour chaque référence une occurence dans le codage de la table est recherchée. La construction des entrées non indifférentes qu'impose cette occurence est entreprise dans la sous-table dont elle est le titre. Le même module décrit par la table 6 : EC est ainsi utilisé pour la construction des entrées conditions, actions ou ruptures de séquence.

A chaque rupture de séquence correspond une règle. Le numéro de celle-ci est indiqué dans la ligne S. Le numéro de l'entrée qui doit être inscrite dans la règle est indiqué par le terme du codage qui fait référence à l'instruction, si celle-ci est une action ou une rupture de séquence.

Dans le cas d'une condition, la ligne S permet de séparer chacun des opérandes de la condition de la façon suivante :

Soit I la position dans le codage de la référence à la condition. Le premier opérande est une sous-table dont le titre est en I + 1 et la position de son dernier terme peut être lue sur la ligne S : I1 S (I+1). Les règles dont les ruptures de séquence se trouvent dans le codage entre I+1 et I1 admettent la première entrée de la condition. Le deuxième opérande se trouve codé de la même façon dans les positions comprises entre I1 + 1 et S (I1+1). Ainsi de proche en proche la construction des entrées non indifférentes induites par une des occurences dans le codage de la condition étudiée.

Dans tous les cas, actions ruptures de séquence ou conditions, une ou plusieurs autre intervention de la référence dans le codage met en jeu le même module, de nouveau. En fin de construction le texte de la souche est copié sur la ligne si la page éditée est la première de la table. La ligne est finalement éditée avec son numéro d'ordre dans la table.

8.6.2.4 - Tables de documentation

Table 1 EDIT-TAB

EDITION DU TITRE	*
PREMIERE REGLE EDITEE L1 = 1	*
GO TO CONS-PATRON	*

Table 2 SUITE-DE-TABLE

L1 ← L2	*
VERS TABLE SUIVANTE	*

Table 3 CONS-PATRON

DETERMINATION DE LA DERNIERE REGLE L2	*
BOLLEEN DE SOUCHE CONDITION J = 1	*
CONSTRUCTION D'UN INTERLIGNE	*
OUVERTURE DE LA TABLE	*
PREMIERE REFERENCE	*
A PARTIR DE I = 1 JUSQU'A NT RECHERCHER UNE OCCURENCE DE LA REFERENCE DANS LE CODAGE	*
VERS TABLE SUIVANTE	*

Table 4 EDIT Edition d'une ligne

LA PAGE EDITEE EST LA SUITE D'UNE TABLE	Ø	Ø	Ø	N	N	N
LA REFERENCE EST LA DERNIERE DE LA TABLE	Ø	Ø	N	Ø	Ø	N
TOUTES LES REGLES SONT EDITEES	Ø	N		Ø	N	
CONSTRUIRE LE TEXTE DE SOUCHE				*	*	*
EDITER LA LIGNE. INTERLIGNE	*	*	*	*	*	*
AJOUTER 1 A LA REFERENCE			*			*
FERMER LA TABLE	*	*		*	*	
A PARTIR DE I=1 JUSQU'A NT RECHERCHER UNE OCCURENCE DE LA REFERENCE DANS LE CODAGE			*			*
GO TO EDIT			*			*
GO TO SUITE-DE-TABLE		*			*	
GO TO INIT (TABLE SUIVANTE)	*			*		

Table 5 — R-C — Entrées d'une ligne

LE TERME I EST LA REFERENCE CHERCHEE	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	N
C'EST LA 1° ACTION DE LA TABLE	Ø	Ø	N	N	N	N	N	N	N	N	
C'EST UNE RUPTURE DE SEQUENCE			Ø	Ø	N	N	N	N	N	N	
C'EST LA PREMIERE RUPTURE DE SEQUENCE			Ø	N							
LE TERME EST EDITE DANS LA SOUS-TABLE EN COURS	Ø	N			Ø	Ø	Ø	N	N	N	
ON EST DANS LA PARTIE CONDITION					Ø	Ø	N	Ø	Ø	N	
SOUS-TABLE R-C1											
TOUTES LES BRANCHES DE LA CONDITION SONT ETUDIEES					Ø	N		Ø	N		
OUVRIR LA PARTIE ACTION J=2	*	*									
OUVRIR UNE NOUVELLE SOUS-TABLE : CHGT		*	*					*	*	*	
I1 DERNIER SUIVANT DE I	*	*			*	*	*	*	*	*	
I1 ← I ; K3=1			*	*							
K3 ← AR (I)	*	*					*			*	
CONSTRUCTION DES BORNES DU 1° OPERANDE DE LA CONDITION					*	*		*	*		
SOUS-TABLE R-C1											
CONSTRUIRE LES ENTREES DE LA LIGNE: EC	*	*	*	*	*	*	*	*	*	*	
K3=K3+1. CONSTRUCTION DE K3° OPERANDE						*			*		
I ← I1	*	*	*	*	*		*	*		*	
EXIT F-RC	*	*	*	*	*		*	*		*	*
GO TO R-C1						*			*		

Table 6 EC Inscription d'une entrée

LE TERME I2 EST UNE RUPTURE DE SEQUENCE	Ø	Ø	Ø	N
LA REGLE CORRESPONDANTE EST L1	Ø	N	N	
LA REGLE CORRESPONDANTE EST L2		Ø	N	
CONSTRUIRE LE NUMERO DE COLONNE K1 = LS + (S (I2) - L1) * 4			*	
ECRIRE LA K3° ENTREE EN K1 sq.			*	
EXIT EC	*		*	*
GO TO EDIT		*		

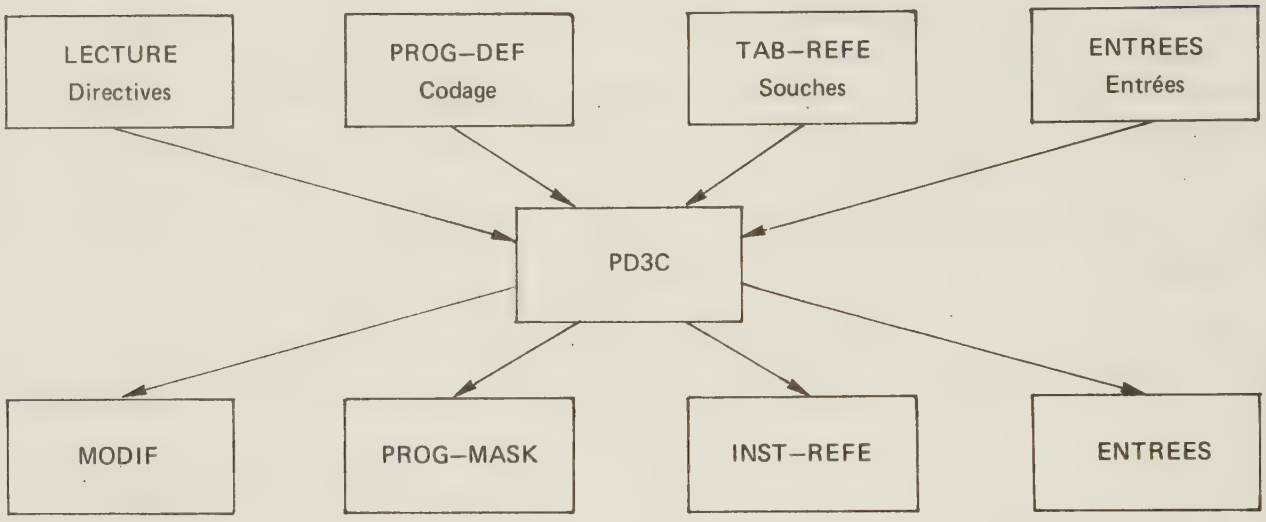
Table 7 CHGT Titre ou changement de sous-table

LE TERME I EST UNE RUPTURE DE SEQUENCE	Ø	N	N
ST (I) EST DANS LA PILE DES SOUS-TABLES TITREES		Ø	N
TITRE DES RUPTURES DE SEQUENCE	*		
TITRE D'UNE SOUS-TABLE ORDINAIRE			*
TITRE D'UNE SOUS-TABLE TITREE		*	
EXIT CHGT	*	*	*

8.6.3 - Directives sémantiques. PD3C

Le programme utilise, comme le précédent, la forme codée des tables de décisions éditée par PD3A, dans les fichiers permanents. Le fichier des directives, lu entièrement, s'applique ainsi aux tables qui ont fait l'objet de l'édition à la phase 3 B. Le programme traite les directives sémantiques et réalise sur le fichier MODIF une image codée des directives de structure. Le traitement est effectué table par table. Les phrases de code, les textes de souche et les entrées sont, après l'étude de chacune, recopiées dans les fichiers du cycle intermédiaire.

Un schéma résumé du fonctionnement du programme peut être :



Les procédures du programme sont réparties en 8 modules différents :

1) Lecture des directives
2) Lecture d'une table
3) Directive COND
4) Directive ACT
5) Directive SUP
6) Edition des images codées des directives de structure
7) Edition du cycle intermédiaire de codage d'une table
8) Recopie sans modification d'une table

8.6.3.1 - Lecture des directives

Les deux fichiers PROG-DEF, du codage des tables et LECTURE, fichier des directives, sont organisés autour de la même unité : la table. Le recalage des phrases de code des tables au cours de la lecture séquentielle du fichier des directives obéit à une structure de double boucle qui traduit l'organisation des fichiers.

La lecture d'une phrase de PROG-DEF est soumise à la présence d'une directive sémantique appliquée à la table correspondante dans le fichier des

directives. La table de documentation 1 suivante explicite la gestion des appels aux procédures décrites dans la suite.

8.6.3.2 - Lecture d'une table

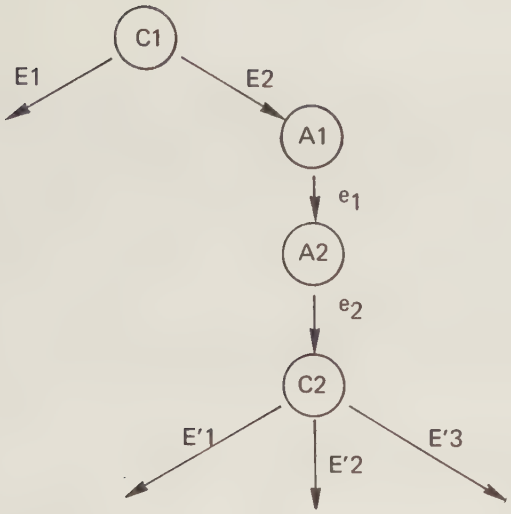
Ce module charge une table en mémoire centrale. Son fonctionnement a été indiqué en *8.6.1.1.* Les lignes S et ST, des derniers suivants et numéro de sous-tables d'édition sont également construites.

Le traitement des directives COND suppose que le codage des arborescences caractéristiques qui seront trouvées soit normal, c'est-à-dire fasse référence aux seules conditions à regrouper. Ce ne pourra être le cas si des actions sont inscrites avant une condition :

TABLE DE DOCUMENTATION

Table 1 LIRE-DIRECTIVES Gestion des appels aux modules de traitement

IL RESTE UNE DIRECTIVE A LIRE	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	N	N
CARTE DE TITRE DE TABLE	Ø	Ø	N	N	N	N	N	N	N	N	N	N		
UNE TABLE EST EN MEMOIRE	Ø	N		Ø	Ø	N	Ø	Ø	N	O	N		Ø	N
DIRECTIVE DE STRUCTURE			Ø	N	N	N	N	N	N	N	N	N		
DIRECTIVE COND				Ø	Ø	Ø	N	N	N	N	N	N		
DIRECTIVE ACT							Ø	Ø	O	N	N	N		
DIRECTIVE SUP										O	O	N		
LES ACTIONS SOUS 6 ONT ETE DEJA DECALEES				Ø	N		Ø	N						
LIRE LA CARTE SUIVANTE	*	*	*	*	*	*	*	*	*	*	*	*		
EDITER LA TABLE EN MEMOIRE													*	
APPELER 1000 FOIS LE MODULE DE RECOPIE													*	*
APPELER LE MODULE D'EDITION	*													
APPELER N FOIS LE MODULE DE RECOPIE (JUSQU'A LA TABLE INDIQUEE EXCL.)	*	*												
APPELER LE MODULE DE CONSTRUCTION			*											
LIRE UNE TABLE						*			*		*			
TRAITER LA DIRECTIVE SUP											*			
DECALER LES ACTIONS SOUS 6					*	*		*	*	*	*			
TRAITER LA DIRECTIVE COND				*	*	*								
TRAITER LA DIRECTIVE ACT							*	*	*					
ERREUR												*		
GO TO LIRE-DIRECTIVES	*	*	*	*	*	*	*	*	*	*	*	*		
STOP RUN													*	*



A la condition peut correspondre une sous-table d'édition, la condition est alors pointée 4 et elle ne pourra être regroupée à des conditions précédent les actions qui n'appartiennent pas à la même sous-table d'édition.

Au contraire, si la condition est pointée 6, les actions précédentes n'interviennent pas dans son évaluation. Pour éviter qu'elles interviennent dans une arborescence caractéristique, elles sont systématiquement décalées, après la condition, sur chaque branche qui en descend. La phrase de code est modifiée de façon correspondante. Ce décalage est exécuté après traitement de l'éventuelle directive SUP. Nous ne décrirons pas l'algorithme élémentaire qui réalise ce décalage.

8.6.3.3 - Directive COND

L'interprétation de la directive COND a été présentée au chapitre 5. En pratique, on peut distinguer quatre phases successives :

- Analyse de la directive,
- recherche dans la phrase d'une arborescence caractéristique,
- construction du codage normal,
- construction de la nouvelle phrase.

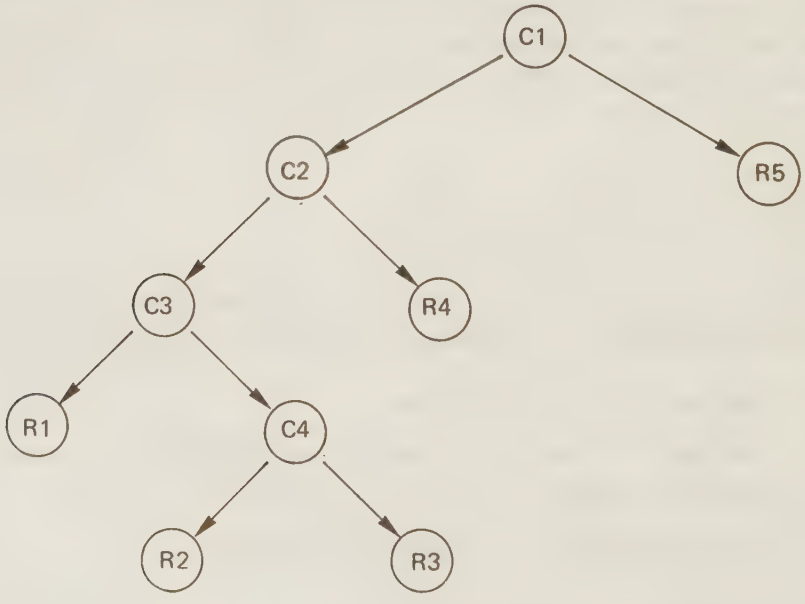
Analyse de la directive

Sous la ligne A qui contient la phrase de code de la table étudiée, une ligne Ø est créée, elle permettra de pointer les termes de la directive. Elle est initialisée à 1. Les références des conditions inscrites dans la directive sont recherchées dans toute la phrase. Une référence trouvée est pointée 2 dans Ø.

Le texte de souche de la condition regroupée est construit dans l'enregistrement du fichier INST-REFE. Les entrées sont séparées et mises en mémoire. Les enregistrements d'entrées sont chaînés entre eux, dans le cas où le nombre d'entrées précisées est supérieur au nombre d'emplacements prévus dans un enregistrement. Le pointeur de chaîne contenu dans chaque enregistrement peut être positif et dans ce cas, il indique le nombre d'emplacements utilisés ; s'il est négatif, sa valeur absolue fournit la clé d'accès à l'enregistrement qui contient les entrées supplémentaires. Le nombre d'entrées proposées dans la directive est le degré extérieur de la future condition regroupée.

Recherche des racines

Les numéros des règles de chacun des n ensembles de règles complémentaires sont construits un à un. Rappelons qu'à chaque règle correspond une rupture de séquence. On recherche dans le codage le premier aïeul Uj de la rupture de séquence, dont l'antécédent soit une condition pointée 2. La référence correspondante est marquée dans Ø J + 2 (3, 4 ...) dans l'ordre des entrées attachées à l'arc incident intérieurement à la racine Uj. La table 2 suivante précise l'algorithme de recherche d'une racine. Il permet de décrire de droite à gauche à partir de la rupture de séquence qui définit la règle, la phrase de code. Un cas, présenté par la règle 4, peut être trouvé dans l'exemple suivant. Soit le jeu de condition (C2, C3) à regrouper. La directive précise l'ensemble supposé complémentaire des règles R1, R2, R3 :



La racine U_2 associé à la règle R2 est C4 qui sera marquéee4 dans Ø. Au cours de la recherche de la racine U_3 on trouvera le sommet C4 déjà pointé. Les ensembles de règles complémentaires corrects sont : R1, R2, R4 ou R1, R3, R4.

A la fin de l'analyse de la directive, les arborescences caractéristiques, formées des conditions pointées et des racines des règles qui viennent d'être déterminées sont construites.

Table 2 — SEC12 — Racines U_j

LA RUPTURE DE SEQUENCE QUI DEFINIT LA REGLE EST UN SUIVANT DU TERME I2	Ø	Ø	Ø	Ø	N
LE POINTEUR DU TERME I2 INDIQUE UNE CONDITION	Ø	Ø	N	N	
ELLE EST POINTEE 2 DANS Ø	Ø	N			
LE TERME Ø (I2) VAUT 1			Ø	N	
RAPPELER I2 EN I5		*	*		
MARQUER J+2 EN O (I5) : DERNIER SUIVANT RENCONTRE DE LA CONDITION	*				
ERREUR				*	
EXIT SEC12	*	*	*		*
GO TO LIRE-DIRECTIVES				*	

Construction du codage normal.

Pour chaque sous-table caractéristique, il reste :

- à déterminer la position K1 de son titre V,

- à vérifier que toutes les conditions du jeu ont une occurence dans la sous-table ; dans le cas contraire, on posera la conclusion que la sous-table de titre V est une sous-table parallèle qui contient des occurences des conditions à regrouper. Elle ne doit pas contenir de racines Uj pointées dans la ligne Ø.

- à vérifier que toutes les branches descendantes de V contiennent une racine et une seule. Dans le cas contraire l'ensemble de règles n'est pas complet.

- enfin, il reste à déterminer si le codage de l'arborescence caractéristique de la liaison est normal, c'est-à-dire s'il ne contient pas d'autres conditions que celles du jeu. La table 3 suivante résume l'organisation de ces vérifications ; K1 décrit successivement toutes les positions de la phrase de code.

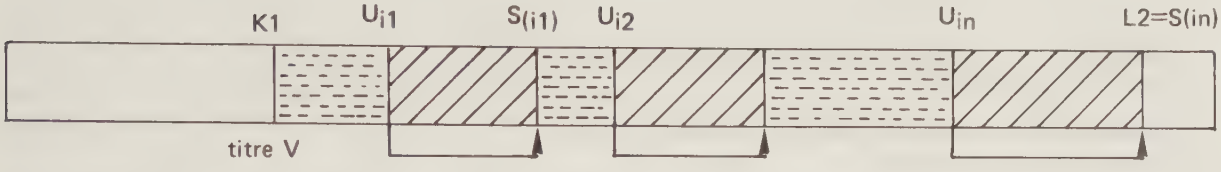
L'algorithme qui permet de faire sortir du codage une condition hors jeu est particulièrement délicat. Il s'agit de recopier en changeant leurs places les différentes branches descendant de la condition hors jeu et des conditions du jeu qui lui sont antécédentes. La phrase de code est allongée, la table devient alors réductible.

Table 3 — SEC4 Construction de l'arborescence caractéristique. Vérifications.

K1 SUPERIEUR AU NOMBRE DE TERMES	Ø	N	N	N	N	N	N	N	N	N	N	N
LE TERME K1 EST UNE CONDITION DU JEU		Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	N
TOUTES LES CONDITIONS DU JEU ONT UNE OCCURENCE ENTRE K1 ET S (K1)		Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	N	N	
IL Y A UN TERME Ø > 2 ENTRE K1 ET S (K1)										Ø	N	
SOUS-TABLE SEC45												
K3 EST SUPERIEUR A S(K1)		Ø	Ø	N	N	N	N	N	N			
LE CODAGE EST NORMAL (I NUL)		Ø	N				Ø	N				
LE TERME K3 EST UNE CONDITION DU JEU				Ø	N	N	N	N	N			
LE TERME K3 EST UNE RACINE U_j (Ø > 2)					Ø	N	N	N	N			
LE TERME K3 EST UNE RUPTURE DE SEQUENCE						Ø	N	N	N			
LE TERME K3 EST UNE CONDITION HORS JEU (Ø = 1 et pointeur = 7)							Ø	Ø	N			
METTRE S (K1) DANS K1											*	
AJOUTER 1 A K1											*	*
ERREUR										*		
INITIALISER K3 A K1. ZERO DANS I		*	*	*	*	*	*	*	*			
SOUS-TABLE SEC45												
RAPPELER K3 DANS PILE (J)					*							
LE CODAGE NE SERA PAS NORMAL RAPPELER K3 DANS I : POSITION D'UNE CONDITION HORS JEU							*					
METTRE S(K3) DANS K3					*							
AJOUTER 1 A K3				*	*		*	*				
ERREUR						*			*			
APPELER LE MODULE DE TRANSFORMATION DE LA PHRASE POUR LA CONDITION HORS JEU			*									
AJOUTER 1 A K1			*									
GO TO SEC4			*								*	*
GO TO SEC45				*	*		*	*				
GO TO SEC6 (TABLE 4)		*										
GO TO LIRE-DIRECTIVES	*					*			*	*		

Construction de la nouvelle phrase.

Pour décrire cette transformation, nous utiliserons le schéma d'une phrase :



dans laquelle les positions du titre V et des racines Uj ont été reconnues. Les zones hachurées délimitent les branches pendantes de la nouvelle condition regroupée. Elles ne sont pas dans l'ordre qui a été précisé dans la directive. Les zones pointillées forment le codage de l'arborescence caractéristique ; elles doivent disparaître.

La référence au texte de la nouvelle condition est mise en K1. Puis les branches pendantes sont recopiées dans l'ordre de la directive à partir de K1. Une précaution est prise de façon à éviter l'écrasement d'une branche non encore recopiée. Enfin la dernière partie de la phrase de code est décalée vers la gauche pour faire suite au nouveau codage. Le traitement de la sous-table V se termine par une reconstruction des lignes S et ST. Si la directive précise plusieurs ensembles de règles complémentaires, le contrôle est passé au module SEC4 à la recherche d'une autre sous-table caractéristique des conditions.

Il faut remarquer que l'ordre des règles a pu être modifié si les règles ne sont pas indiquées par ordre croissant dans la directive. Leur nombre a pu augmenter si le codage de l'arborescence caractéristique n'était pas normal. Dans ce cas, certaines des conditions du jeu n'auront pas disparues de la souche condition de la table. Ces transformations risque d'induire des erreurs si plusieurs directives COND sont appliquées sans précaution à la même table.

La table 4 suivante se lit pour IP variant de 1 à N, si N est le nombre de règles de l'ensemble complémentaire, ou encore le nombre d'entrées.

Table 4 — SEC6 — Recopie des branches de la nouvelle condition

IP EST SUPERIEUR AU DEGRE EXTERIEUR N DE LA CONDITION	Ø	Ø	N	N	N
LA DIRECTIVE PRECISE UN AUTRE ENSEMBLE DE REGLES COMPLEMENTAIRES	Ø	N			
LA BRANCHE IP PEUT ETRE RECOPIEE DANS LES POSITIONS AVANT I1			Ø	N	N
IL RESTE DE LA PLACE DANS LE TAMPON POUR Y RECOPIER LA BRANCHE I1				Ø	N
DECALER A GAUCHE LE RESTE DE LA PHRASE	*	*			
RECHARGER LES LIGNES S ET ST	*	*			
S (K1) dans K1	*				
SOIT I1 LA POSITION DE LA BRANCHE NON RECOPIEE LA PLUS A GAUCHE			*	*	*
RECOPIER LA BRANCHE I1 DANS LE TAMPON RAPPELER CE TRANSFERT				*	
RECOPIER LA BRANCHE IP DANS LES POSITIONS SUCCESSIVES DE LA LIGNE A			*	*	
ERREUR BUFFER PLEIN					*
EFFACER LA PHRASE					*
AJOUTER 1 A IP			*	*	
GO TO SEC6			*	*	
GO TO SEC4	*				
GO TO LIRE-DIRECTIVES		*			*

8.6.3.4 - Directive ACT

Le traitement de la directive ACT se décompose en 2 modules :

- analyse de la directive et pointage des actions et des règles,
- construction du codage.

La directive ACT peut remplir deux rôles : résumer sur une seule ligne des actions entreprises en séquences sur une ou plusieurs règles, résumer sur une seule ligne des actions exécutées chacune sur des règles différentes et qui ont des éléments en commun.

Les algorithmes mis en jeu dans le traitement d'une directive ACT sont élémentaires. Au cours de l'analyse de la directive, les textes de souches et d'entrées sont construits dans les fichiers respectifs INST-REFE et ANTREES. Les règles sont pointées dans la ligne Ø à la position de leurs ruptures de séquence, par le numéro d'ordre (augmenté de 2) de l'entrée qui leur correspond. Enfin les actions à regrouper sont pointées 2 dans la ligne Ø.

La deuxième phase de construction du nouveau codage consiste à supprimer une succession de références d'actions pointées, pour les remplacer par la seule référence à la nouvelle action. Le numéro de l'entrée trouvé dans la ligne Ø sous une des ruptures qui terminent la séquence, est rappelé dans le terme. Les numéros d'entrée doivent être identiques pour toutes les ruptures de la séquence qui peut être conditionnelle (contenir une condition pointée 4). Il y a impression d'un diagnostic dans le cas contraire.

8.6.3.5 - Directive SUP

La ligne ST qui indique sous chaque référence le numéro d'ordre de la sous-table d'édition correspondante a été construite lors de la lecture de la phrase de code. Le traitement de la directive SUP se résume donc à construire les numéros des sous-tables qui doivent être supprimées. Deux cas peuvent se présenter.

1) Il s'agit d'une sous-table non titrée à l'édition. Son titre est une condition pointée 4 ou 5. Le pointeur devient alors 6 ou 7 respectivement.

2) Il s'agit d'une sous-table titrée. Son titre est une référence au nom, pointée 1. Elle est supprimée.

Après traitement de la directive entière, la ligne ST est reconstruite.

8.6.3.6 - Construction des images codées des directives de structure

Les directives de structure sont transmises au programme PD3D qui les traite, par l'intermédiaire du fichier MODIF. L'organisation de ce fichier et la forme codée retenue pour chaque directive seront décrites avec le programme PD3D.

8.6.3.7 - Edition du cycle intermédiaire de codage

La lecture d'une table s'est accompagnée de la recherche des références minimum et maximum aux textes de souche. Les textes générés à l'occasion du traitement des directives COND et ACT ont été affectés à des références supérieures au maximum précédent. NOR mesure le nombre de textes ainsi ajoutés à la table, NA1 rappelle le nombre de textes ajoutés à toutes les tables précédentes.

Les textes de la table non modifiés sont donc recopiés un à un dans le fichier INST-REFE ; ils subissent une translation. Les références sont modifiées dans la phrase. Pour chaque texte les enregistrements d'entrées sont également recopiés. Enfin la phrase elle-même est éditée dans le fichier PROG-MASK. NOR est ajouté à NA1 puis remis à zéro.

8.6.3.8 - Recopie d'une table sans modification

Les décalages de textes qui viennent d'être indiqués imposent d'exécuter une translation sur toutes les références d'une table qui ne subit par ailleurs aucune directive sémantique.

8.6.4 - Programme PD3D - Directives de structure

Ce programme traite les directives de structures et modifie le codage des tables de décisions de façon à regrouper ou séparer les tables comme demandé par les directives :

AJRS

SEP

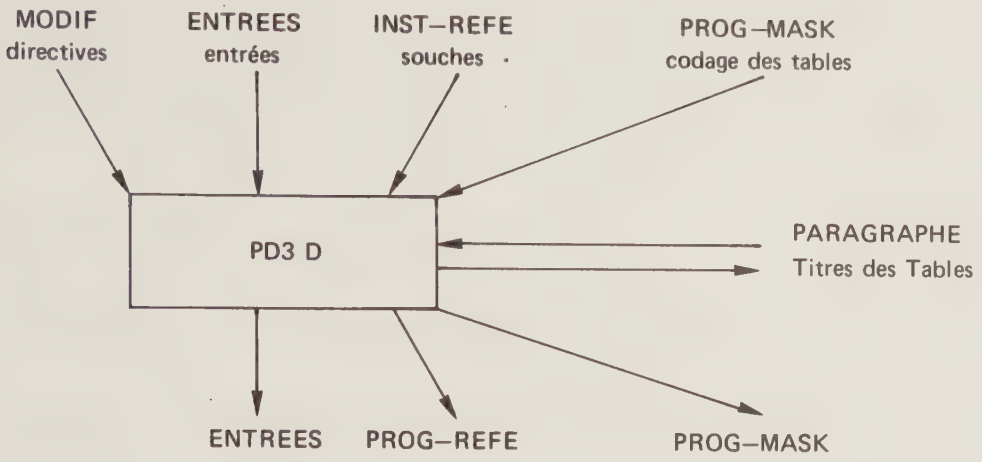
REG

NUL

CHGT

Les deux modules fonctionnels principaux sont ceux qui codent les algorithmes de séparation d'une sous-table et de regroupement de deux tables. Ces algorithmes ont été étudiés en détail au chapitre 4. Les fichiers de souches et d'entrées sont recopiés dans les fichiers du nouveau cycle. Les seules nouvelles instructions éventuellement générées par PD3D sont les ruptures de séquence nécessaires aux liaisons entre les nouvelles tables créées. Enfin le fichier des titres de tables est évidemment remis à jour. Le schéma de fonction-

nement de ce programme est précisé par la figure suivante :



La PROCEDURE DIVISION de ce programme est organisée autour de quatre modules principaux :

I- Chargement du codage de toutes les tables

II- Détermination du type des tables - Directives SEP AJRS

III- Construction et édition du nouveau codage - Traitement des directives REG

IV- Mise à jour de la table à adressage dispersé contenant les titres de tables

Nous étudierons successivement chacun de ces quatre modules.

8.6.4.1 - Module I - Chargement du codage des tables

Ce chargeur est semblable à celui que l'on trouve au début du programme PD2A. Son rôle est identique : le codage des tables est chargé en mémoire centrale de façon à ce que le codage de chaque table soit accessible à tout instant. La place de chaque tête de table est mémorisée de deux façons différentes. Tout d'abord en séquence dans le tableau SUITE, d'autre part, dans le tableau LIEU à la ligne correspondante à l'adresse de hash-code du nom de la table concernée. Il est ainsi possible de répondre rapidement aux deux questions suivantes :

- où est le début du codage de la $N^{ième}$ table ?
- où est le début du codage de la table dont le nom est donné ?

8.6.4.2 - Module II

8.6.4.2.1 - Détermination du type des tables

A chaque table est associé un nombre caractéristique des actions qui doivent être entreprises sur son codage. Ce code se trouve à la suite du nom de la table dans le fichier PARAGRAPHE : variable PREM-REF.

PREM-REF vaut de	1	à 203 si le nom correspondant doit être modifié, il indique alors l'adresse du nouveau nom.
" "	999	si le nom correspondant est le titre édité d'une sous-table.
" "	1000	si la table correspondante doit être supprimée à l'édition et donc dans le codage.
" "	1001	si la table correspondante doit être éditée et n'être regroupée sur aucune autre.
" "	1002	si la table doit être regroupée sur toute celles qui l'appellent, avec un nouveau titre. Cette valeur se trouve en face du nouveau nom de la table. Après regroupement la table doit être supprimée : ceci est obtenu par une directive REG,T titre suivi d'une directive NUL.
" "	1003	comme 1002 mais la table est conservée et éditée avec son nouveau titre. Il n'y a pas de directive NUL.
" "	1004	La table correspondante doit être regroupée avec son titre sur toutes celles qui l'appellent. Elle est ensuite effacée des fichiers : directives REG,T. et NUL.
" "	1005	Même signification que 1004, mais la table est conservée et éditée.
" "	1006	La table correspondante doit être regroupée sans titre sur celles qui l'appellent. Elle est ensuite supprimée, directives REG,S. et NUL.
" "	1007	Même signification que 1006. La table est conservée.

8.6.4.2.2 - Fichier MODIF

La détermination du type de chaque table est réalisée au cours de la lecture du fichier MODIF qui contient un codage des directives de structure. Celles-ci sont hiérarchisées en trois groupes :

1° groupe directive AJRS.

2° groupe directives REG et SEP

3° groupe directives NUL et CHGT.

Un enregistrement du fichier MODIF contient les deux variables COD et DIG numériques et la zone TEZ permettant de transmettre 30 caractères alphanumériques. Les exclusions entre directives de structure signalées au chapitre 7 imposent qu'une table ne peut faire l'objet que d'une seule directive dans chaque groupe. A chaque table sont associés 4 enregistrements dans le fichier MODIF.

Le premier contient l'annonce des groupes utilisés par les directives de la table. COD vaut zéro. Et DIG est construit à l'aide des numéros des groupes.

Le second enregistrement contient un codage de la directive du premier groupe, s'il y en a une. Dans ce cas COD vaut 1, DIG n'est pas utilisé et TEZ résume les caractères alphanumériques non blancs de la directive. Dans le cas contraire l'enregistrement n'est pas utilisé.

Le troisième enregistrement contient un codage d'une directive du deuxième groupe, s'il en est une présente. Pour une directive SEP, COD vaut 5 ou 6 (option titre) le numéro de la sous-table est mémorisé par DIG, le titre s'il existe est en TEZ. Dans le cas d'une directive REG, COD vaut 2 (option S), 3 (option T) ou 4 (option T avec changement de titre).

Enfin le quatrième enregistrement est réservé aux éventuelles directives du troisième groupe. COD vaut 7 pour une directive NUL et 8 pour une directive CHGT. Le nouveau nom de la table est aloes en TEZ.

8.6.4.2.3 - Directive AJRS

Le traitement complet des directives AJRS et SEP est entrepris au cours de la lecture pas à pas du fichier MODIF. Rappelons que les directives REG sont traduites en termes de type de table au cours de cette même lecture. La directive AJRS est la première directive de structure que subit une table. Il s'agit de permettre à l'édition la présence d'une ou plusieurs sous-tables supplémentaires. Les titres de ces sous-tables ne peuvent être que des condi-

tions désignées par le numéro de la ligne où elles sont éditées. Pratiquement le traitement consiste à ajouter l'indication d'une rupture sémantique sur le pointeur des conditions référencées. Le numéro d'une condition est d'abord construit puis son occurence est recherchées dans la table enfin le pointeur est modifié.

8.6.4.2.4 - Directive SEP

Cette directive a pour objet d'imposer l'édition en tables distinctes d'une table et de l'une de ses sous-tables. Nous avons signalé au chapitre 4 l'essentiel de l'algorithme de séparation. Le traitement de cette directive est réalisé en deux étapes :

- Recherche du titre de la sous-table à séparer. Cette recherche consiste à reconstruire l'ordre d'édition des sous-tables.

- Construction du pointeur du titre de la nouvelle table. Ce pointeur peut en effet être modifié suivant les cas d'espèces, d'après le tableau :

ancien pointeur	nouveau pointeur
2	3
3	3
4	7
5	7
6	7
7	7
8	8 si la table est conditionnelle
9	9 sinon

L'instruction de liaison avec la nouvelle table est ensuite générée. Enfin le codage est recopié après la dernière table. Les éventuelles instructions de ruptures de séquence : VERS LA TABLE SUIVANTE sont remplacées par des instructions de ruptures de séquence faisant référence au nom de la table.

Si la directive SEP a l'option T, c'est-à-dire si la sous-table à séparer n'est pas titrée, le titre est attribué à la table créée et une instruction de séquence est générée dans la table principale.

A la fin de la génération de la nouvelle table, la recherche d'un autre titre d'une sous-table parallèle et de même ordre est entreprise dans le codage de la table principale. Si l'une est trouvée, un nouveau nom de table est créé à partir de l'ancien. Cette recherche supplémentaire n'est entreprise que si l'option T est utilisée dans la directive SEP.

8.6.4.3 - Module III - Elaboration du nouveau codage

Ce module est organisé autour de l'algorithme de regroupement de tables de décisions. Nous aurons à décrire le cas particulier du regroupement d'une table appelée par l'intermédiaire du verbe PERFORM.

Le codage des tables est lu en séquence à l'aide du tableau SUITE construit lors du chargement. Chaque terme est analysé :

L'instruction référencée est lue dans le fichier INST-REFE d'entrée, une condition ou une action autre que PERFORM est recopiée dans le fichier des souches sortie. Les entrées correspondantes sont également recopiées.

La rencontre d'une rupture de séquence GO TO ou d'une instruction du type VERS LA TABLE SUIVANTE entraîne la recherche du nom de la table en cause. Son adresse est déterminée. Un tableau de clé mémorise les sous-tables titrées déjà rencontrées au cours de la lecture. Si la table appelée se trouve être une des sous-tables titrées, l'instruction d'appel est simplement recopiée et le traitement standard repris. Sinon le type de la table appelée est lu dans le fichier PARAGRAPHE. Les cas où la table appelée ne doit pas être groupée sont encore éliminés, le libellé de l'instruction est éventuellement changé si le nom de la table appelée est modifié. Enfin dans tous les autres cas, le regroupement est entrepris. Si la sous-table doit être titrée un terme supplémentaire est édité dans le codage. Il est de la forme :

1 00 adresse (4 chiffres)

L'adresse est celle du nom de la sous-table dans le fichier PARAGRAPHE.

8.6.4.3.1 - Liaison par PERFORM

Nous étudierons dans ce paragraphe le regroupement de deux tables liées par un verbe PERFORM. Cette liaison est en effet très particulière et conduit à des difficultés, un même module pouvant à la fois être appelé par l'intermédiaire d'un verbe PERFORM et par une rupture simple GO TO. L'interprétation de l'instruction EXIT, existante dans le programme ou générée pendant le codage des tables atomiques, est différente suivant la nature de la liaison d'appel.

Le regroupement de deux tables liées par PERFORM impose de mémoriser dans une pile particulière PILPE la position dans le codage de la table qui appelle, de façon à reprendre sa description en séquence. Il faut encore rappeler le paragraphe fin de module appelé par l'instruction PERFORM, sous la forme de l'adresse de son nom, dans le tableau noté MEMO.

La détermination du premier pointeur et la description de la table appelée ne sont pas distinguées du traitement ordinaire.

Les instructions EXIT générées dans les tables atomiques font référence au nom du paragraphe dans lesquelles elles se trouvent. Ainsi la rencontre d'une instruction EXIT dans le codage d'une table, suffit pour déterminer un nom de paragraphe fin de module. L'adresse de ce nom est calculée et recherchée dans le tableau MEMO. La position correspondante dans PILPE permet la reprise de la description de la table contenant l'inscription PERFORM. Cette reprise est accompagnée de l'empilement ordinaire dans la pile de l'instruction EXIT. A la fin de la table appelante, la description de la table appelée est introduite sans présenter de cas particulier.

Nous résumerons cette description croisée des deux tables par l'exemple du schéma :

```
                              I1
                              ↓
T1  ...  A  ...  PERFORM T2 THRU T3  ...  B  0000
                          I2
                          ↓
T2  ...  C  ...  GO TO T3  ...  D  ...  0000
                          I3
                          ↓
T3  ...  E  ...  EXIT  T3  ...  F  ...  0000
```

en indiquant les opérations et le contenu des tableaux PILE, PILPE et MEMO.

a) description de A

b) PILPE : I1 ; MEMO : T3

c) description de C

d) PILE : I2 ; PILPE : I1 ; MEMO : T3

e) description de E

f) recherche dans MEMO de T3 d'où I1 dans PILPE
PILE : I2 I3 ; PILPE et MEMO vides

g) description de B

h) en fin de table I3 est dépilé ; PILE : I2

i) description de F

j) en fin de table I2 est dépilé, PILE est vide

k) description de D

l) en fin de table la pile est vide : FIN

Le résultat est formé de A C E B F D

Si à la rencontre d'une instruction EXIT le nom du paragraphe associé n'est pas trouvé dans le tableau MEMO, il n'y a pas eu d'appel du module par une instruction PERFORM et le terme correspondant est simplement recopié sans modification.

TABLES DE DOCUMENTATION

Table 1 MODULE I Mise en mémoire des tables

IL RESTE DE LA PLACE POUR METTRE UN TERME EN MEMOIRE	Ø	Ø	Ø	N
TOUT LE PROGRAMME EST LU	Ø	N	N	
LE TERME INDIQUE LE NOM DE LA TABLE		Ø	N	
LIRE UN TERME		*	*	
LE METTRE EN MEMOIRE		*	*	
SIGNALER LE DEPASSEMENT				*
METTRE EN MEMOIRE LA POSITION DU TERME DANS LES TABLEAUX SUITE ET LIEU		*		
GO TO MODULE I		*	*	
GO TO MODULE II	*			
FIN				*

Table 2 MODULE II Lecture du fichier MODIF
Directive AJRS

TOUTES LES TABLES ONT ETE ETUDIEES	Ø	N	N	N	N
IL Y A DES DIRECTIVES APPLIQUEES A LA TABLE		Ø	Ø	Ø	N
IL Y A UNE DIRECTIVE AJRS		Ø	Ø	N	
L'ENREGISTREMENT 2 DE MODIF CONTIENT LA DIRECTIVE AJRS		Ø	N		
LIRE L'ENREGISTREMENT 1 DE MODIF		*	*	*	*
CONSTRUIRE J1 J2 J3 CARACTERISTIQUES DES GROUPES DE DIRECTIVES PRESENTS		*	*	*	*
CONSTRUIRE LA REFERENCE MINIMUM DE LA TABLE		*			
LIRE LES NUMEROS DES CONDITIONS A SINGULARISER		*			
RECHERCHER LES OCCURENCES DES CONDITIONS		*			
SIGNALER L'ERREUR INTERNE			*		
LE TYPE DE LA TABLE EST 1001					*
CONSTRUIRE LES REFERENCES D'APPEL DE LA TABLE SUIVANTE					*
GO TO MODULE II					*
GO TO MODULE III	*				
VERS LA TABLE SUIVANTE		*		*	
FIN			*		

Table 3 TEST-SEP directives des groupes 2 & 3

IL Y A UNE DIRECTIVE DU GROUPE 2	2	2	2	3	3	3	4	4	4	5	5	6	6	N	N	N
2 REG, S 3 REG, T 4 REG, T TITRE 5 SEP 6 SEP, T N PAS DE DIRECTIVES																
IL Y A UNE DIRECTIVE DU GROUPE 3	3	4	N	3	4	N	3	4	N	4	A	4	A	3	4	N
A → 3 NUL 4 CHGT N PAS DE DIRECTIVES																
CONSTRUIRE LE NOM DE LA NOUVELLE TABLE LE STOKER LE TYPE EN EST 1001												*	*			
CONSTRUIRE LE NOUVEAU NOM DE LA TABLE CHAINER LE FICHIER.NEW-T		*					*	*	*			*			*	
FAIRE LE TRAITEMENT DE LA DIRECTIVE SEP TT-SEP										*	*	*	*			
LE TYPE DE LA TABLE EST 1000 + n	6	7	7	4	5	5	2	3	3	1	1	1	1	0	1	1
CONSTRUIRE LES REFERENCES D'APPEL DE LA TABLE SUIVANTE	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
GO TO MODULE II	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

Table 4 TT-SEP Traitement d'une directive SEP

IL Y A UNE SOUS-TABLE DU DEGRE VOULU	Ø	Ø	Ø	Ø	N
LA SOUS-TABLE EST TITREE	Ø	Ø	N	N	
LA DIRECTIVE EST SEP, T	Ø	N	Ø	N	
RECHERCHER LA TETE DE LA SOUS-TABLE DE NUMERO VOULU	*	*	*	*	
SIGNALER UNE ERREUR DE DIRECTIVE	*			*	
OUVRIR LA TABLE APRES LE DERNIER TERME MIS EN MEMOIRE		*	*		
CREER UNE INSTRUCTION GO TO EQUIVALENTE A VERS TABLE SUIVANTE		*	*		
DETERMINER LE PREMIER POINTEUR DE LA TABLE		*	*		
RECOPIER LA SOUS-TABLE REMPLACER SES TERMES DANS LA TABLE PRINCIPALE PAR 9 99 9999		*	*		
FERMER LA NOUVELLE TABLE		*	*		
GENERER DANS LA TABLE PRINCIPALE L'INSTRUCTION DE LIAISON AVEC LA NOUVELLE TABLE		*	*		
GO TO TT-SEP			*		
EXIT TT-SEP	*	*		*	*

Table 5 MODULE III Construction du nouveau codage
Regroupement de tables

TOUS LES TERMES DU CODAGE ONT ETE DECRITS	Ø	N		
LE TERME I DU CODAGE EST UN NOM DE PARAGRAPHE		Ø	Ø	N
LA TABLE CORRESPONDANTE DOIT ETRE SUPPRIMEE		Ø	N	
EDITER LES 3 TERMES D'OUVERTURE DE LA TABLE			*	
INITIALISER LES CLES DE SOUS-TABLES TITREES			*	
AJOUTER 1 A I			*	*
RECHERCHE LA FIN DE LA TABLE		*		
GO TO MODULE III		*		*
GO TO MODULE IV	*			
VERS TABLE SUIVANTE			*	

Table 6 REC-NC Analyse d'un terme

le pointeur du terme i indique												
UNE CONDITION	Ø	N	N	N	N	N	N	N	N	N	N	N
UN NOM DE SOUS-TABLE TITREE		Ø	N	N	N	N	N	N	N	N	N	N
UNE FIN DE TABLE			Ø	Ø	N	N	N	N	N	N	N	N
UNE RUPTURE DE SEQUENCE					Ø	Ø	Ø	Ø	Ø	N	N	Ø
LE VERBE DE L'ACTION EST PERFORM										Ø	N	
LE VERBE DE LA RUPTURE EST STOP					Ø	N	N	N	N			N
LE VERBE DE LA RUPTURE EST EXIT						Ø	Ø	N	N			N
LE NOM DU PARAG EST DANS MEMO						Ø	N					
L'INSTRUCTION EST GO TO								Ø	Ø			N
LE PARAGRAPHE REFERENCE EST SOUS CLE								Ø	N			
LA PILE EST VIDE			Ø	N								
AJOUTER 1 A I												
DETERMINER LA POSITION DE LA TABLE SUIVANTE DANS LE CODAGE												*
CONSTRUIRE LES NOMS ET ADRESSES DES 2 PARAGRAPHES DEBUT ET FIN										*		
DETERMINER LA POSITION DE LA TABLE APPELEE									*	*		
DETERMINER LA POSITION DU PERFORM DE LA TABLE APPELANTE						*						
DETERMINER LE PREMIER POINTEUR						*						
METTRE 2 DANS LA CLE DU NOM		*										
SOIT I LE SOMMET DE LA PILE				*								
EDITER LE TERME	*	*			*		*	*			*	
EDITER LE TEXTE ET LES ENTREES	*				*		*	*			*	
FERMER LA TABLE AJOUTER 1 A I			*									
GO TO MODULE III			*									
GO TO REC-NC	*	*		*	*	*	*	*				
GO TO R1									*	*	*	*

Table 7 R1 Elaborer la décision de regroupement

le nom de la table a changé	Ø	N	N	N	N	N	N
LE TYPE DE LA TABLE INDIQUE							
UN CHANGEMENT DE NOM SANS REGROUPEMENT		Ø	N	N	N	N	N
UN REGROUPEMENT SIMPLE			Ø	Ø	N	N	N
UN REGROUPEMENT AVEC TITRE					Ø	Ø	N
L'APPEL EST PAR PERFORM			Ø	N	Ø	N	
LIRE LE NOM DE LA TABLE A REGROUPER	*	*	*	*	*	*	*
CONSTRUIRE L'ADRESSE DU NOUVEAU NOM	*						
GENERER LA NOUVELLE INSTRUCTION DE RUPTURE DE SEQUENCE		*					
EDITER LE TERME INDIQUANT LE NOM DE LA SOUS-TABLE					*	*	
METTRE DANS MEMO L'ADRESSE DU NOM DU PARAGRAPHE FIN ET DANS PILPE LA POSITION DE L'INSTRUCTION PERFORM			*		*		
DETERMINER LE PREMIER POINTEUR DE LA NOUVELLE TABLE			*	*	*	*	
SOUSTRAIRE 1 DE I			*	*	*	*	
EDITER LE TERME		*					*
EDITER LE TEXTE ET LES ENTREES DU TERME		*					*
GO TO R1	*						
GO TO REC-NC		*	*	*	*	*	*

ANNEXE I

ÉTUDE BIBLIOGRAPHIQUE - LES TABLES DE DÉCISIONS

A I - a. NOTICE BILIOGRAPHIQUE (page 219)

A I - b. ETUDE BIBLIOGRAPHIQUE (page 226)

A I - a. NOTICE BIBLIOGRAPHIQUE

A I - a 1. Tables de décisions - Ouvrages généraux

1) G. Baglin et J. Klee — Edition moderne d'entreprise

Les tables de décisions - Initiation pratique

Cet excellent ouvrage didactique présente de façon claire et progressive les notions utiles à la maîtrise de l'outil de programmation que sont les tables de décisions. Il contient une initiation au processeur I.B.M. : Decisions Logic Translator, implémenté sur la série 360, et dont le langage cible est FORTRAN.

2) S. L. Pollack, H. T. Hicks, W. J. Harrison — John Wiley and sons, Inc.

Decisions tables - Theory and pratice

Ce livre a été signalé au chapitre 2. Il est sans doute appelé à devenir l'ouvrage de référence sur le sujet. Les auteurs y reprennent les concepts qu'ils avaient présentés dans de nombreux articles au cours des années précédentes.

3) S. L. Pollack — R.M.-3669-PR Rand Corp, Santa Monica, Californie, mai 1963

Analysis of the décision rules in decision tables

4) id. — Comm. ACM 8,11 nov. 1965

Conversion of limited entry decision tables to computer programs.

Cet article est résumé en A I - b.

5) id. — Data processing Vol. 8 1965 pp 485 - 492

Decision tables for system design

6) id. Comm. ACM 14,1 janv. 1971 P52.

Comment on the conversion of decisions tables to computer programs

Cette courte lettre introduit la necessité des entrées conditions * et $

7) id. et Sprague Comm. ACM 9,5 May 1966 pp 319 - 320

On storage space of decision tables

Ces deux lettres de réponse sont signalées dans la deuxième partie de l'annexe.

Il faut encore citer les articles de présentation générale :

8) N. Chapin DPMA Quaterly 3,3 April 1967 pp 2 - 23

An introduction to decision tables

9) CODASYL Systems Group and the JUG of ACM 116 pages 20-21 Sept. 1962

Proceedings of the decision tables symposium

Cet ouvrage ancien a le mérite d'être le premier à présenter de façon complète la pratique des tables de décisions. Il a permis de définir les besoins que doivent couvrir le concept de tables de décisions. Il a en outre fixé la forme d'édition des tables de décisions.

10) J. N. Cuvelette Informatique et gestion mars 1973 pp 101 - 106

La traduction des tables de décisions

Cet article présente une discussion simple et complète des avantages et inconvénients de l'emploi des tables de décisions comme outil d'analyse et de documentation d'un programme de gestion.

Nous conclurons cette première liste en citant :

11) *SIGPLAN Notices* Sept. 1971

relatif aux tables de décisions. L'un des articles de ce volume a été résumé dans la suite de cette annexe. L'ouvrage contient une bibliographie importante dans laquelle nous avons largement puisé.

A I - a 2. Traduction des tables en programme

Le problème de la conversion des tables de décisions fait l'objet de la plupart des publications relatives aux tables de décisions. Nous citerons :

12) L.I. Press Comm. ACM 8,6 juin 1965 pp 385 - 390

Conversion of decision tables to computer programs

L'article est résumé dans la deuxième partie de l'annexe.

13) Reinwald and Soland J. ACM 13,3 July 1966 pp 339 - 358

Conversion of limited entry decision tables to optimal computer programs

I Minimum average processing times

14) id. J. ACM 14,4 Oct. 1967 pp 69 - 73

II Minimum storage requirements

15) K. Schwayder Comm. ACM 14,2 Feb. 1971 pp 69 - 73

Conversion of limited entry decision tables to computer programs: a proposed modification to Pollack's algorithm.

Cet article est résumé dans la suite.

16) Muthukrishnan and Rajaraman — Comm. ACM 13,6 juin 1970 pp 347 - 351

On the conversion of decision tables to computer programs

17) N. Chapin — Comm. ACM 10,8 Aug. 1967 pp 507 - 512

Parsing on decision tables

Il faut encore signaler les publications du CODASYL Systems group et toutes celles qui présentent tel ou tel préprocesseur de tables de décisions. On trouvera de très nombreuses références et une présentation de tous les processeurs existants dans les deux ouvrages suivants :

18) H. McDaniel — Brandon/Systems Presse, Princeton 1970

Decision tables software

19) id. — id.

Application of decision tables

Nous citerons pour terminer cette liste les quatre articles suivants :

20) J. H. King — Comm. ACM 9,11 nov. 1966 pp 796 - 801

Conversion of decision tables by rule mask techniques

L'article est résumé dans la deuxième partie de l'annexe.

21) M. Verhelst — Comm. ACM 15,11 nov. 1972 pp 974 - 980

The conversion of limited entry decision tables to optimal and near-optimal flow-charts : two new algorithms

22) J. H. King — Comm. ACM 11,10 Oct. 1968 pp 680 - 684

Ambiguity in limited entry decisions tables

Résumé dans la suite de l'annexe.

23) Kirk — Comm. ACM 8,1 janv. 1965 pp 41 - 43

Use of decisions tables in computer programming

Cet article présente une technique d'utilisation différente des précédentes. Chaque table est interprétée à l'exécution du programme sans être traduite en un programme.

A I - a 3. Recette des programmes

Les ouvrages parus sur ce très vaste sujet sont évidemment nombreux. Nous nous limiterons à citer ceux qui ont donné lieu à des résultats pratiques. Il faut remarquer que les tables de décisions ne sont nulle part présentées comme un outil de recette. Des articles non cités ici, et qui traitent de l'analyse des systèmes logiques à l'aide de tables concluent généralement que les tables, en permettant une analyse précise et documentée, sont le meilleur garant d'un programme correct et efficace.

24) SEMA — Contrat CRI - SEMA N° 70020 avril 1971

Etude sur la simplification d'ordinogrammes appliquée à la conception de machines à logique séquentielle synchrone et à la programmation sur ordinateur

Ce rapport signale l'utilisation des tables de décisions, mais introduit l'étude d'un programme par son organigramme.

25) SESA — Contrat CRI - SESA N° 70021 sept. 1971 et 01 Informatique mensuel juin 1972

Un générateur automatique de fichier de test

26) Contrat D.R.M.E. - Université de Toulouse N° 69-34-358

Analyse, Contrôle, Vérification, Correction des programmes par Graphes

Il s'agit de quatre rapports successifs. La vérification d'un programme est là encore réalisée en étudiant le graphe du programme, mis en mémoire à l'aide d'une matrice de connexion.

27) C.A.R. Hoare Comm. ACM 14,1 janv. 1971 pp 39 - 45

Proof of a program : procedure FIND

La procédure FIND est une procédure de tri dans un tableau de nombres. L'auteur décrit formellement, dans cet article la preuve de la procédure.

28) E.W. Dijkstra BIT Vol. 8 1968 pp 174 - 186

A constructive approach to the problem of program corectness

29) M.M. Burstall Computer J. 1968 pp 41 - 48

Proving properties of programs by structural induction

30) R.L. London BIT Vol. 10 1970 pp 168 - 182

Proving programs correct : some techniques and examples

A I - a 4. Documentation automatique

Nous signalerons enfin les études liées au tracé automatique d'ordinogrammes d'un programme donné. Le but de telles études peut être à la fois de documentation et de vérification ou débourrage d'un programme. La documentation obtenue n'est dans tous les cas utilisables que par l'analyste/programmeur.

31) Knuth — Comm. ACM 6,9 sept. 1963

Computer drawn flowcharts

32) Main — Proceedings 20th ACM Notl Conf. Août 65

Automatic flowcharts desing

33) Goetz — Applied data research, Inc. ; Princeton 1965

Automated program documentation. Autoflow.

34) Sherman — Comm. ACM 9,12 Dec. 1966

Flow trace, a computer programm for flowcharting computer programs

Nous citerons pour terminer les réalisations étudiées par ECA Automation dans le domaine du débourrage et de la documentation des programmes.

35) ECA Automation — Contrat DRME N° 70/432 mars 1971

Le système STEP. Aide à la mise au point des programmes Outil évolué de débourrage et de test des programmes, particulièrement des programmes temps réels.

36) ECA Automation Tisserand et Guiard — Congrès AFCET nov. 1972 Vol. 2 pp 311 sq

Outil de Documentation Automatique des Symboles d'un Système. Cet article décrit un outil de documentation automatique des variables d'un programme.

A I - b. ETUDE BIBLIOGRAPHIQUE

Nous suivrons, pour présenter les résumés d'articles annoncés, le plan qui a servi dans la première partie de l'annexe.

Les tables de décisions ont été utilisées d'abord pour analyser et documenter un algorithme dont la logique complexe ne peut s'exprimer dans un autre des moyens classiques de documentation. Très rapidement le problème s'est ensuite posé de rendre ces tables accessibles directement à la machine. La plus grande partie des articles techniques parus sur le sujet des tables de décisions s'intéressent à la conversion des tables de décisions en programmes. Nous résumerons ici les articles qui présentent les résultats les plus originaux. L'arbitraire est entré pour beaucoup dans ce choix.

Les problèmes soulevés lors de l'analyse d'un algorithme et de la construction des tables de décisions ont été beaucoup moins largement traités. Quelques ouvrages généraux ou didactiques existent. Nous avons rendu compte de l'un d'entre eux dans le chapitre 2 ; un second, français, est signalé dans la notice bibliographique 1.

Aucune étude, à notre connaissance, n'a cherché à obtenir une documentation, de façon automatique ou non, des programmes écrits dans un langage évolué, à l'aide de tables de décisions.

Enfin le problème de la vérification et de la recette des programmes de gestion à l'aide de tables a fait l'objet d'une étude menée par la SESA et résumée dans la dernière partie.

A I - b 1. Conversion des tables de décisions en programmes

Nous résumerons successivement les articles signalés sous les références

23
20
12
4
7
15
11
22

Utilisation des tables de décisions en programmation

H. W. Kirk — Comm. ACM Janv. 1965 8,1

L'article propose une technique de transformation des tables de décisions qui les rend accessibles à l'ordinateur. Au contraire de l'organigramme, les tables de décisions sont donc capables de communiquer directement avec l'ordinateur, elles ont de plus l'avantage d'être plus lisibles que lui et de rendre mieux compte des situations logiques complexes en précisant de façon plus synthétique les relations entre variables et traitements.

L'auteur précise sur un exemple de tables à entrées limitées, ce qu'il est convenu d'appeler, entrées conditions, entrées actions et règle de décisions. La technique utilisée pour l'emploi en ordinateur de telles tables est liée à l'interprétation d'une forme codée de la table de décisions.

Les entrées de la table sont remplacées en mémoire centrale par deux matrices binaires. La matrice de table, D, formée de vecteurs règles, recopie les entrées conditions ; Oui est remplacé par 1, Non et l'entrée indifférente par zéro. La matrice masque M qui permet de lever les indéterminations liées aux zéros de la matrice précédente recopie les entrées conditions non indifférentes : Ø ou N deviennent 1.

L'interprétation du vecteur donnée, formé de 1 et de 0 suivant que la condition correspondante est vérifiée ou non, consiste à étudier les 2 matrices colonne après colonne. On réalise le produit logique d'une règle du masque et du vecteur donnée, le résultat est comparé à la colonne correspondante de la matrice D. S'il y a égalité la règle vérifiée est trouvée sinon on étudie la colonne suivante des matrices.

Ainsi la table :

C1	Ø	Ø	-	N	
C2	Ø	-	N	-	
C3	N	Ø	N	-	
	R1	R2	R3	R4	Autre

conduit aux deux matrices :

$$M = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Pour des valeurs données nous supposerons que C1 est vraie, C2 et C3 sont fausses. Le vecteur donné est donc :

$$V = \begin{matrix} 1 \\ 0 \\ 0 \end{matrix}$$

Les produits logiques du vecteur V et des colonnes successives de M conduit aux colonnes :

$$\begin{matrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

Chaque colonne est ensuite comparée à la colonne correspondante de D. La règle 3 est satisfaite.

Deux améliorations peuvent être apportées à cette technique. Tout d'abord les deux matrices binaires peuvent être condensées en une seule matrice dont l'élément soit un caractère. La comparaison du vecteur donnée aux positions non blanches de la matrice est alors directement exécutée. D'autre part, le temps de recherche de la règle vérifiée peut être réduit si l'on regroupe dans une première matrice les règles ayant le plus d'entrées indifférentes ; puis dans les matrices successives les règles en ordre croissant d'entrées imposées. Il faut remarquer que ce gain de temps s'accompagne d'une perte de place en mémoire centrale.

La conclusion de l'auteur est à l'avantage de cette technique qui permet de regrouper en une matrice toute la logique d'un algorithme. La programmation assurée de façon automatique est rapide et sans redondances.

Conversion de tables de décisions en programmes

J. H. King Comm. ACM nov. 1966 9,11 pp 796 - 801

Dans l'article précédent KIRK présentait la technique des matrices masques sous sa forme la plus simple. Il signale la faiblesse de l'algorithme quant à l'efficacité des programmes créés et propose d'y remédier en divisant les matrices pour prendre en compte de façon primaire la fréquence des règles.

Il s'agit de construire deux matrices M et D ; l'une sépare les entrées indifférentes des entrées précisées, l'autre les Ø des N. la détermination d'une règle impose de rechercher d'abord la valeur de chacune des conditions puis de comparer le vecteur ainsi construit à chacune des colonnes de M et D. Les règles peuvent être rangées en ordre décroissant de fréquences pour accélérer en probabilité le temps moyen de choix d'une règle. Il reste que toutes les conditions doivent être évaluées, même si la valeur de plusieurs se révèle être indifférente à la règle finalement satisfaite.

Procédure proposée

Nous reprendrons l'exemple cité dans le résumé précédent pour présenter la méthode de "recherche de la règle par masque interrompu". Supposons que R3 soit la règle la plus fréquente et C1 une condition assez longue à évaluer. La méthode de Kirk impose de rechercher la valeur de C1 à chaque fois que la règle R3 est satisfaite, alors que cette condition y est indifférente.

La méthode proposée consiste dans ce cas à évaluer d'abord C2 et C3 et de rechercher si la règle 3 est vérifiée, ensuite seulement d'évaluer C1 pour construire la règle 1, 2 ou 4 vérifiée. Cette stratégie peut être condensée par la donnée du tableau suivant, ou tableau d'arrangement.

2	3	0	1	0	0	0
0	0	3	0	1	2	4

La première ligne s'intéresse aux conditions de la table et indique l'ordre dans lequel elles sont évaluées. Chaque groupe est séparé par des 0. La seconde ligne référence de la même manière les règles de la table. Le tableau se lit de gauche à droite. Un zéro sur une ligne fait basculer la lecture sur l'autre ligne.

La stratégie de Kirk serait pour notre exemple :

1	2	3	0	0	0	0
0	0	0	1	2	3	4

La technique ainsi décrite conduit à un programme plus efficace mais plus complexe et donc plus volumineux. L'auteur montre qu'il s'agit essentiellement de mettre en mémoire le tableau décrit ci-dessus et un vecteur d'index qui permet de le décrire. Pour estimer le temps d'exécution du programme objet et construire le meilleur arrangement, il faut connaître la fréquence relative des règles et les temps d'évaluation des conditions.

Soit donc donné :

$$f = \begin{bmatrix} f1 \\ . \\ . \\ . \\ fn \end{bmatrix} \qquad t = \begin{bmatrix} t1 \\ . \\ . \\ . \\ tm \end{bmatrix}$$

pour n règles et la règle autre et m conditions. Avec :

$$\sum_{i=1}^{n} \left(f_i \right) + f_{autre} = 1$$

Pour un tableau d'arrangement donné, on construit la matrice A dont la i° colonne indique par un 1 les conditions qui ont été évaluées avant l'essai de la règle i. Les 1 de A répètent au moins les 1 de la matrice M.

s note le temps nécessaire à l'essai d'une règle. Soit v le vecteur :

$$v = \begin{bmatrix} 1 \\ 2 \\ . \\ . \\ . \\ n \end{bmatrix}$$

et g la ligne des f_i dans l'ordre d'essai des règles :

$$g = (f_{i_1}, f_{i_2}, \ldots, f_{i_n})$$

Le temps total d'exécution d'un essai, en probabilité, pour un arrangement choisi s'écrit :

$$T = t^t * A * f + s\ g\ v + f_{autre} * \left(\sum_{i=1}^{m} t_i + ns \right)$$

Il s'agit de choisir un arrangement tel que :

$$V = t^t A f + s\ g\ v$$

diminue. Le dernier terme introduit par la règle autre, est indépendant de l'arrangement choisi.

L'auteur propose quatre stratégies différentes pour construire l'arrangement utile :

Etant donné un ordre d'évaluation des conditions, il est décidé d'essayer une règle dès que possible. Pour construire un ordre des conditions on évalue le vecteur :

$$P = M * f$$

Un élément p_j s'interprète comme la probabilité qu'à la condition j d'être utile à l'essai de la règle satisfaite, c'est-à-dire que l'entrée ne soit pas indifférente.

La stratégie A consiste à choisir les conditions en ordre de probabilité p_j décroissante.

La stratégie B conduit à évaluer les conditions dans l'ordre croissant des temps t_i nécessaires à leur évaluation.

Il est également possible de choisir un ordre d'essai des règles et de décider l'évaluation d'une condition quand elle est nécessaire.

La stratégie C choisit les règles en ordre décroissant de fréquence.

Pour tenir compte des temps d'évaluation des conditions, on formera pour chaque règle le temps d_i somme des temps d'évaluation t_j des conditions qui ont une entrée significative sur la règle. Les règles peuvent alors être choisies en ordre décroissant des rapports f_i / d_i. C'est la stratégie D.

L'article se termine par une comparaison des temps obtenus par les quatre stratégies précédentes et l'algorithme de Kirk. L'auteur a construit cinq exemples. Suivant les cas, les gains de temps vont de 74 % à une perte de 9 % du temps moyen, par rapport à l'algorithme de Kirk.

Conversion des tables de décisions en programmes

Comm. ACM 8,6 juin 1965 L.I. Press

Trois critères d'optimisation se présentent pour mesurer l'efficacité de la traduction d'une table en un programme.

- La place prise en mémoire par le programme objet. Elle sera en fait mesurée par le nombre de conditions qu'il faut évaluer.

- Le temps d'exécution du programme objet. On doit alors disposer d'une donnée supplémentaire : la fréquence de chaque règle à l'exécution. Le temps d'exécution sera alors exprimé par un temps moyen en probabilité.

- Le temps de compilation ou de traduction de la table en programme.

Deux techniques peuvent être mises en jeu pour traduire une table. Les articles précédents et l'un des algorithmes ci-dessous s'intéressent à des vecteurs et matrices qui recopient en la masquant l'information contenue dans la table. Une autre technique consiste à traduire la partie condition de la table en un organigramme. Tous les articles suivants s'intéressent à cette dernière technique. Les algorithmes qui y sont décrits ont en commun la même ossature ; ils ne diffèrent que par le critère de choix de la phase 1.

Nous décrirons cette organisation commune, une seule fois, d'après l'exposé qui en est fait dans l'article résumé ici.

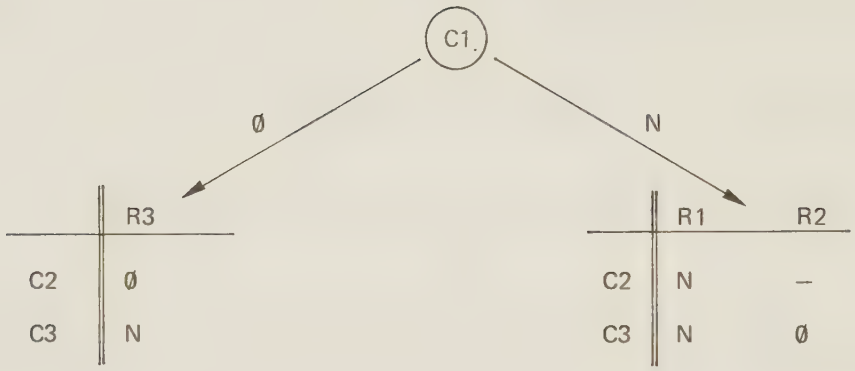
Soit l'exemple de la table :

	R1	R2	R3	Autre
C1	N	N	Ø	
C2	N	-	Ø	
C3	N	Ø	N	

1) La première phase consiste à choisir une condition, qui sera la première de l'organigramme. Soit C1.

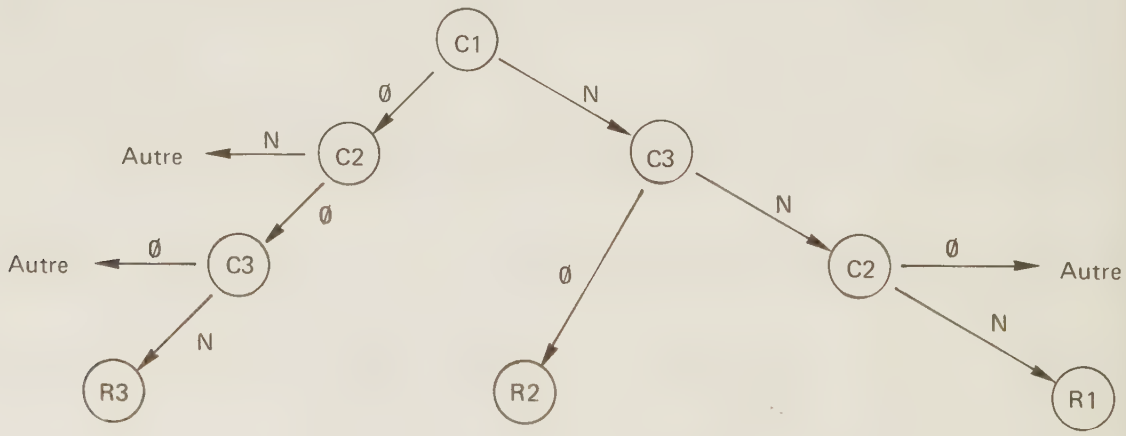
2) La table est alors séparée en deux sous-tables. A la branche Ø descendant de la condition sont attachées les règles qui contiennent Ø ou - sur la ligne de la condition. A la branche N celles qui contiennent N ou -.

On obtient ici :



3) L'algorithme est repris en 1 pour les sous-tables qui contiennent encore plusieurs règles.

4) Il reste alors à séparer pour chaque branche pendante la règle de la règle autre. Dans l'exemple :



Il reste à préciser comment choisir une condition pivot en 1.

L'auteur propose pour diminuer le nombre de points de test de l'organigramme de choisir d'abord les conditions qui ont le moins d'entrées indifférentes. En cas de conflit, s'il existe une ligne qui ne contienne que des Ø ou que des N, la choisir, sinon on cherchera à ce qu'une des sous-tables créées ait une ligne de Ø ou une ligne de N.

Pour cela, on distingue les entrées positives (Ø ou -) des entrées négatives (N ou -) d'une ligne C_i. La ligne C_i est dite complémentaire de C_j si les règles positives de C_i ont la seule entrée Ø (ou la seule entrée N) sur C_j et/ou si les règles négatives de C_i ont la seule entrée Ø (N) sur C_j. Par exemple :

C_i	Ø	-	N
C_j	Ø	Ø	N

et C_{ij} vaut ici 2, il mesure le nombre de règles qui assurent la complémentarité, C_{ji} vaut 1 (règle 3).

On choisira alors la condition qui maximise CS_i :

$$CS_i = \sum_{j=1}^{m} (C_{ij}) \qquad j \neq i$$

Méthode par masque

La méthode précédente présente l'inconvénient de construire plus de points de test qu'il n'y a de conditions dans la table, d'où une perte de place en mémoire. La méthode suivante conduit à un programme plus concis. Le temps d'exécution est sensiblement plus long. Au contraire le temps de compilation, en produisant un programme plus court, est plus bref.

o Phase 1. Evaluer chacune des n conditions pour des valeurs données et construire un tableau (n, 2) dans lequel :

(1, 0) représente "vrai"
et (0, 1) représente "faux"

o Phase 2. Les entrées de chaque règle sont remplacées par deux bits :

1, 0 pour Oui
0, 1 pour Non
1, 1 pour l'entrée indifférente.

On compare la matrice donnée à chacune des règles successives pour déterminer celle qui est vérifiée.

Conversion des tables de décisions en programmes

S. L. Pollack Comm. ACM 8,11 nov. 1965

Dans son introduction, l'auteur rappelle la construction de deux préprocesseurs de conversion de tables de décisions en programmes COBOL : l'un est écrit par le groupe de travail N° 2 du SIGPLAN (ACM Los Angeles), l'autre par le Census Bureau.

Deux algorithmes sont présentés, L'un minimise la place prise en mémoire, l'autre le temps de calcul du programme objet. Ils s'appliquent à des tables de décisions à entrées limitées sans redondances ni contradictions.

Ces algorithmes utilisent la méthode de séparation en sous-tables développée dans l'article précédent. Nous ne détaillerons que le critère de choix de la condition pivot.

1er algorithme

- Déterminer les conditions qui ont un DC minimum (dash-count)
  - Pour chaque règle le compte de colonne est 2^r, si r est le nombre de tirets dans cette colonne,
  - Le DC d'une ligne est la somme des comptes de colonnes qui ont un tiret sur cette ligne,
- En cas de conflit choisir la ligne qui a un Δ maximum.
  - Le compte de $\emptyset$ d'une ligne est la somme des comptes des colonnes qui ont un $\emptyset$ sur cette ligne.
  - Définition identique pour le compte de N.
  - Δ est la valeur absolue de la différence.

2ème algorithme

Le but est de minimiser le temps d'exécution du programme résultant. Il faut pour cela une donnée supplémentaire à savoir les fréquences relatives de chaque règle. La règle Autre doit rester relativement peu fréquente.

On définit de même :

- Le compte de colonne 2^r, r est le nombre de tirets.
- Le DC pondéré ; WDC de chaque ligne somme des produits : fréquence * compte des colonnes pour les colonnes qui ont un tiret sur la ligne,

- Le DC garde la même définition,
- Le Δ également.

On choisit la ligne qui a un WDC minimum. En cas de conflit choisir la ligne de D minimum. Si des conflits subsistent, choisir la colonne de DC minimum. Ceci ne diminue pas le temps d'exécution, mais peut diminuer la place prise en mémoire.

Algorithme 394

R. B. Dial Comm. ACM 13,9 sept. 1970 pp 571 - 572

Cet article court propose le listing ALGOL d'un programme qui code l'algorithme numéro 2 de Pollack décrit dans l'article précédent. Le résultat de l'exécution est une arborescence mise sous forme d'un dictionnaire des suivants à 3 colonnes.

La première colonne indique un numéro de référence des conditions. La seconde, le numéro de la ligne qui code le suivant Oui de la condition ou le numéro d'une règle, sommet pendant de l'arborescence. (Le nombre est alors négatif.) La troisième colonne signale suivant le même code le suivant Non.

Nous nous permettrons de critiquer cette représentation en mémoire d'une table à entrées limitées. Le programme proposé transforme une forme matricielle entière :

- 1, 0, 1 pour respectivement N - et Ø

en une autre forme matricielle qu'il faudra ensuite interpréter pour obtenir une exécution ou un programme d'exécution. L'intermédiaire du dictionnaire des suivants du graphe formé par l'arborescence de Pollack est inutile. En effet, l'algorithme de Pollack fournit conditions et règles dans l'ordre précis qui est le leur dans une phrase de la forme :

IF ... THEN .... ELSE ...

avec les imbrications autorisées en COBOL. Pratiquement, il aurait été plus simple de construire directement le code à générer ou un codage de l'arborescence qui s'en rapproche à l'aide des numéros de référence aux conditions ou aux règles. Nous avons détaillé une telle forme de représentation des tables de décisions à entrées limitées au chapitre 4.

Sur la place prise en mémoire par une table de décisions

V. G. Sprague Comm. ACM 9,4 mai 1966

Cette courte lettre à l'éditeur contient deux remarques sur l'algorithme 1 de Pollack.

La première est une interprétation du DC et du Δ dans le plan de KARNAUGH. Le mérite en est d'établir un parallèle entre la réduction des fonctions booléennes et la réduction de l'organigramme équivalent à une table de décisions. Ce qui amène l'auteur à la conclusion que la solution de notre problème doit traîner dans quelque revue d'électronique. Retenons que le DC est la surface occupée par la règle dans le plan de Karnaugh.

La seconde remarque reproche à ces algorithmes de s'intéresser exclusivement aux entrées conditions des tables de décisions, à l'exclusion des souches et entrées actions. Le critère de place minimum en mémoire est ainsi grossièrement traduit par : nombre de sommets conditions minimum dans l'organigramme. La place prise par les actions des règles qui sont répétées sur différentes branches pendantes est négligée.

Conversion de tables de décisions en programmes

Une modification à l'algorithme de Pollack

K. Shwayder Comm. ACM 14,2 fév. 1971 pp 69 - 73

Une introduction résume le deuxième algorithme de Pollack. L'auteur détaille ensuite sur un exemple un parallèle entre la théorie de l'information (codage par dichotomie de Shannon), et le problème de la conversion d'une table en un ordinogramme, en cherchant à minimiser le temps d'exécution.

- Une table de décisions possède une certaine quantité d'information. L'entropie d'une source dont les règles seraient les messages.

- L'ordinogramme obtenu est formé de conditions qui peuvent prendre deux valeurs. Il peut être considéré comme le codage des messages. Ce codage a une certaine efficacité que Shannon a mesurée.

- Le problème de la traduction des entrées conditions d'une table, de façon à réduire le temps d'exécution est ainsi identique au problème du codage des messages à travers un canal sans bruit.

- L'efficacité du codage peut alors se mesurer.

De ces principes, l'auteur déduit les deux critères de choix suivants :

Algorithme 3

La table doit être à entrées limitées sans ambiguïté et sans règle "Autre". Ce dernier point ne peut pas être considéré comme une restriction. En effet, si la règle Autre a une grande probabilité, il vaut mieux la préciser. L'algorithme consiste à séparer des sous-tables en choisissant des conditions pivots.

On choisira à chaque étape la condition qui apporte le plus d'information sur la table. En cas de conflit, on choisira l'une ou l'autre. L'entropie d'une condition est calculée comme suit :

Pour chaque condition, on évalue :

- P_0 Probabilité que l'entrée ne soit pas indifférente.
- P_1 Probabilité d'un Ø si l'entrée n'est pas indifférente.

- P_2 Probabilité d'un N si l'entrée n'est pas indifférente.

$$P_1 + P_2 = 1$$

L'entropie est alors :

$$-\left(P_1 \text{Log}_2 P_2 + P_2 \text{Log}_2 P_1 \right) . P_0$$

Il faut remarquer que le résultat n'est pas nécessairement optimal. Ceci vient de ce qu'on n'optimise qu'une sous-table à la fois. La recherche à chaque étape de décomposition, de sous-tables identiques peut être programmée. La décomposition s'arrête quand il ne reste plus qu'une règle par sous-table. On ne cherche pas à évaluer les conditions qui peuvent demeurer. La règle autre n'est pas complètement différenciée. Si on attend, de ce fait, un taux d'erreur relativement trop grand, il faut employer la règle suivante.

Algorithme 4

Cet algorithme sépare complètement la règle "autre" cependant son résultat est en général moins bon que celui du précédent. Il conduit à un meilleur arbre que l'algorithme de Pollack.

- Pour chaque colonne compter le nombre d'entrées non indifférentes.

- Diviser la fréquence de la règle par ce nombre. Le résultat est le poids de la colonne.

- Pour chaque ligne, le poids de la ligne est la somme des poids des colonnes qui ont un Ø ou un N sur la ligne.

Choisir la ligne qui a un poids maximum. En cas de conflit, choisir l'une ou l'autre. Le poids de la ligne est une approximation meilleure que le WDC de Pollack, de l'entropie de la condition.

SIGPLAN Notices Septembre 1971 pp 104 - 111

Tashio Yasui - Quelques aspects des techniques de conversion de tables

Les algorithmes de conversion proposés par S. L. POLLACK ou L.I. PRESS font intervenir des critères de bon sens, sans base théorique qui les introduisent.

Notations

Soit n le nombre de conditions intervenant dans une table de décisions à entrées limitées sans redondances ni règle ELSE. Dans l'ensemble des 2^n sommets d'un cube unité construit dans un espace à n dimensions, ou n-cube, on notera un sommet simple par le n-upple de ses coordonnées : (x_1, x_2 ... x_n) où x_i vaut zéro ou un.

Un k-cube est caractérisé par un n-upple, où k des coordonnées x_i ont la valeur - indifférente, les n-k restant valent zéro ou un. A une règle de la table, on associe un k-cube B_i dont les coordonnées recopient les entrées de la règle :

$$x_j^i = 0 , 1 , - \quad \text{pour} \quad N , Y , -$$

A une table complète, on peut associer une partition π du n-cube.

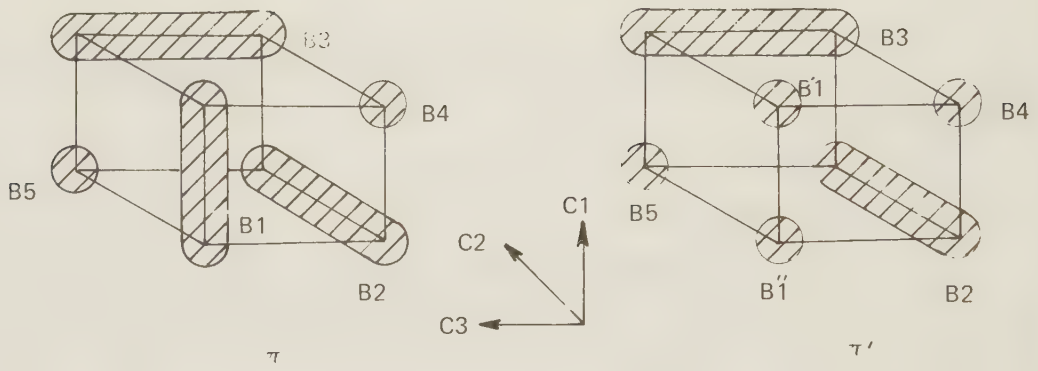
$$\pi = \left\{ B_i \right\} \quad \text{où} \quad B_i \cap B_j = \emptyset \text{ si } i \neq j$$

$$\text{et} \quad \cup B_i = S$$

Soit $d\pi$ le nombre d'éléments de π

Pour deux partitions π_1 et π_2 on notera $\pi_1 \leq \pi_2$ si et seulement si chaque k-cube ou bloc de π_1 est contenu dans un bloc de π_2

Ainsi dans la figure suivante construite pour n = 3 $\pi' < \pi$:



π π'

Un arbre de décisions est un graphe arborescent tel que :

- chaque sommet non terminal est une condition C_i.
- sur un chemin qui lie la racine à un sommet terminal une condition C_i n'apparaît pas plus d'une fois.

Le coût d'un arbre de décisions est le nombre de sommets non pendants, c'est encore le nombre de sommets pendants moins un. Le coût mesure la place prise en mémoire par le codage défini par l'arbre.

A un chemin de l'arbre qui lie la racine à un sommet pendant, on associe un k-cube B_t tel que :

x_i^t = - si C_i n'est pas sur le chemin

x_i^t = 0 (1) si le chemin suit le descendant de gauche (droite) de la condition i.

Les blocs ainsi construits sont disjoints ; leur réunion couvre tous les sommets du n-cube. On dira qu'un arbre de décisions réalise une partition π si et seulement si il y a identité entre les blocs de π et les blocs B_t précédents. L-arbre est alors noté T (π), son coût est d π - 1.

Une partition est réalisable s'il existe au moins un arbre qui la réalise. Ils ont tous le même coût. La partition π de la figure précédente n'est pas réalisable. On trouvera une discussion de ce cas en 3.4.3. La partition π' est réalisable. Le coût de T (π') vaut $d\pi' - 1 = d\pi - 1 + (d\pi' - d\pi)$.

La perte $d\pi' - d\pi$ est nécessaire. Le problème est de trouver une partition π' réalisable telle que $\pi' \leqslant \pi$ et qui minimise la perte. Soit $\ell(\pi) = \min_{\pi'} [d\pi' - d\pi]$ le coût minimal sera $d\pi + \ell(\pi) - 1$.

L'auteur détaille ensuite l'algorithme général qui permet de construire un arbre de décisions pour une partition donnée. C'est une adaptation de l'algorithme général de POLLACK en termes de partitions successives dans un n-cube. Le critère de choix d'une condition qui permet de séparer une partition par deux partitions incluses reste entier.

L'auteur démontre que s'il existe une condition C_i sans perte (c'est-à-dire sans entrée indifférente) pour une partition donnée, le coût d'un arbre de décisions réalisant $\pi' \leqslant \pi$ dont C_i est racine est toujours inférieur ou égal à celui de tout autre arbre.

Par cet intermédiaire, l'auteur introduit le cas du dash-count nul de l'algorithme de POLLACK et propose le critère de choix suivant (I.L.M.) :

A chaque itération :

- s'il existe une condition sans perte la choisir,
- sinon choisir la condition pour laquelle la perte sur la partition en cours est minimum, c'est-à-dire ayant le moins d'entrées indifférentes.

Ce deuxième cas interdit de parler d'un optimum. Le calcul de la perte permet de mesurer l'efficacité de l'arbre obtenu, L'auteur démontre ensuite les propositions suivantes qui permettent de comparer l'algorithme de POLLACK et l'I.L.M. à l'algorithme optimal.

Proposition A

Pour tout $n \geqslant 4$ il existe une partition d'un n-cube pour laquelle le coût de l'arbre de décisions est 2^{n-4} fois inférieur à celui de l'arbre obtenu par l'algorithme de POLLACK ou l'I.L.M.

Proposition B

Pour tout $n \geqslant 5$ il existe une partition d'un n-cube pour laquelle le coût de l'arbre de décisions construit par l'algorithme de POLLACK est 2^{n-4} fois supérieur au coût de l'arbre de décisions obtenu par l'I.L.M.

Proposition C

Pour tout $n \geqslant 5$ il existe une partition d'un n-cube pour laquelle le coût de l'arbre construit par l'I.L.M. est 2^{n-4} fois supérieur à celui obtenu par l'algorithme de POLLACK.

Proposition D

Pour tout algorithme qui choisit d'abord une condition n'ayant pas d'entrée indifférente :

$$\frac{2^n}{8} \leqslant \max \; l(\pi) \leqslant \frac{2^n}{2} \log \frac{n}{2}$$

La borne supérieure résulte d'une hypothèse supplémentaire :

A chaque itération la perte maximum que l'on puisse introduire est :

$$2^{k-1}/k \qquad k \geqslant 3$$

En conclusion, l'auteur signale qu'une démarche très voisine peut être entreprise pour les algorithmes qui cherchent à minimiser le temps moyen d'exécution.

Ambiguïtés dans les tables de décisions à entrées limitées

J. H. King Comm. ACM Oct. 1968 Vol. 11 N° 10 pp 680 - 684

L'axiome de base dans l'étude des tables de décisions est que pour toute valeur des données une règle et une seule est satisfaite. La partie condition de la table, qui permet de séparer les règles sera sans ambiguïtés dans une table bien construite. Au contraire, une table de décisions est dite ambiguë si, pour un ensemble particulier des valeurs des données, les entrées conditions de deux règles, différentes par leurs entrées actions, sont satisfaites.

Il faut distinguer plusieurs cas d'ambiguïtés. Une ambiguïté apparente peut être liée à la présence d'actions identiques dans les règles ambiguës. L'une quelconque des règles ambiguës peut être choisie les mêmes actions sont entreprises. Dans le cas de conditions dépendantes, c'est-à-dire qui ont des opérandes en commun, certaines des combinaisons des valeurs des conditions sont inaccessibles et peuvent donner lieu à une ambiguïté des entrées conditions. Il n'y correspond aucune ambiguïté réelle. Chacun de ces deux cas d'ambiguïté apparente est illustré par un des exemples suivants :

le bit 1 vaut zéro	Y	N	-
le bit 2 vaut zéro	Y	-	N
le résultat doit être zéro	*		
un		*	*

âge < 18	Y	-	N
âge > 65	-	Y	N
GO TO	1	2	3

L'auteur distingue ces cas d'ambiguïté fausse du cas de la réelle ambiguïté qui conduit à une contradiction. Les définitions proposées par S. L. POLLACK de l'ambiguïté et de la contradiction paraissent beaucoup trop restrictives pour l'utilisation pratique. (Deux règles doivent contenir un couple Y, N sur la même ligne.) Pour deux conditions dépendantes POLLACK impose d'écrire des entrées supplémentaires. Ainsi la deuxième table doit être écrite par exemple :

âge < 18	Y	-	N
âge > 65	N	Y	N

Un préprocesseur tel que DETAB-65 qui analyse tous ces cas de fausses ambiguïtés conduit à un codage lourd. Il ne prend en compte que la logique des entrées conditions sans étudier les relations de dépendances entre conditions. Il apparaît ainsi à l'auteur qu'un préprocesseur de tables de décisions ne doit pas analyser systématiquement la nature des conditions et préférer un dialogue avec l'utilisateur dans le cas d'ambiguïté possible :

Une table de décisions présente une ambiguïté possible si la présence ou non d'ambiguïté ne peut être décidée sans étudier la nature des conditions et leurs relations entre elles.

Procédure proposée

En construisant la partie condition de la table, l'analyste indique le minimum d'entrées pour définir correctement les actions du programme. Ainsi, si l'état Oui d'une condition impose que telle autre soit Non, l'entrée de cette dernière sera marquée indifférente. On évite ainsi la programmation d'une condition inutile quant à la logique. Le processeur étudie la possibilité d'ambiguïtés possibles des conditions. L'utilisateur prend lui-même la décision de signaler que les valeurs ainsi données sont logiquement inaccessibles, ou qu'au contraire la table est erronée.

Décrivons l'algorithme qui permet ce résultat :

La partie condition d'une table de décisions à entrées limitées est représentée par une matrice masque M et une matrice de décisions D (voir pages 227/229). Soit m le nombre de conditions et de lignes, n le nombre de règles et de colonnes. m_i et d_i représentent les colonnes de M et D. Le couple $\{m_i , d_i\}$ caractérise l'état des conditions pour que la règle i soit satisfaite. m_i signale les conditions qui doivent intervenir et d_i les valeurs qu'elles doivent avoir. Une entrée indifférente conduit à un zéro dans d_i. Soit $\cup$ et $\cap$ les opérateurs OU inclusif et ET qui opèrent terme à terme sur des vecteurs. A_k représente l'ensemble des actions de la règle k.

Il y a une ambiguïté possible entre R_j et R_k si et seulement si :

$$m_j \cap d_k = m_k \cap d_j \quad \text{avec :}$$

$$A_j \neq A_k$$

Si $A_j = A_k$ il ne peut y avoir ambiguïté au sens indiqué puisque les actions seront les mêmes.

L'auteur montre que $m_j \cap m_k$ caractérise les valeurs des conditions telles que R_j et R_k sont satisfaites à la fois. Dans ce cas, on doit avoir :

$$\left\{ m_j \cap m_k , d_j \right\} = \left\{ m_j \cap m_k , d_k \right\}$$

c'est-à-dire : $m_j \cap m_k \cap d_j = m_j \cap m_k \cap d_k$

mais par construction des vecteurs $m_j \cap d_j = d_j$ et $m_k \cap d_k = d_k$

d'où le résultat annoncé. Le calcul fait est réciproque.

L'article conclut par une présentation des résultats et à l'introduction de cette procédure dans les processeurs de tables DETAB-65 et FORTAB.

A I - b 2. Analyse et construction des tables de décisions

Nous résumerons, ici, d'une part l'article signalé sous la référence 17 qui introduit la notion de hiérarchie entre tables de décisions, d'autre part l'annexe principale du livre déjà signalé Decision tables. Theory and Practice, qui décrit une présentation théorique des tables de décisions.

Séparation de tables de décisions

Ned Chapin Comm. ACM 10,8 août 1967 pp 507 - 512

L'analyse à l'aide de tables de décisions, de problèmes réels non triviaux se heurte à la difficulté soulevée par la dimension des tables utilisées. Des tables de dimensions trop grandes, deviennent impraticables, et les avantages classiques de leur utilisation sont perdus. L'article présente quelques moyens de réduire ou décomposer une table trop grande :

Le premier de ces moyens et il est courant, est l'utilisation de l'entrée indifférente. Du fait que n conditions permettent de distinguer 2^n règles, des règles dont les entrées actions sont identiques peuvent être regroupées en une seule. Certaines entrées conditions de celle-ci seront indifférentes. Cependant, deux règles dont les entrées actions sont identiques ne sont pas toujours ainsi réductibles.

Il peut rester des tables irréductibles de grandes dimensions. Il faudra décomposer une telle table en plusieurs autres, liées entre elles. La dernière action d'une règle indique habituellement de réinterpréter la table avec de nouvelles valeurs des données. Les actions d'une telle table modifient l'état d'au moins une condition, dans le cas contraire, on aboutit à une boucle sans fin. La dernière action d'une règle peut aussi indiquer d'exécuter une autre table. La table ainsi appelée est alors exécutée de façon complètement indépendante. Si une règle de cette seconde table appelle la première, celle-ci est exécutée entièrement, en particulier les conditions sont réévaluées, des actions de la seconde table ont pu en modifier la valeur.

Il existe une deuxième catégorie d'appel de tables. Dans le cours des actions en séquence d'une règle, on peut trouver un appel à une autre table, celle-ci est exécutée de façon indépendante. Certaines règles se terminent par une sortie particulière qui indique de reprendre la description de

la table d'appel. La reprise se fait en séquence dans la règle d'appel sans réévaluer les conditions même si leur état a été modifié par l'exécution de la seconde table.

La séparation d'une table trop grande conduit à de plus petites tables de décisions liées par l'un ou l'autre de ces moyens. Il reste à construire des critères pour séparer les tables de façon à obtenir une documentation efficace qui ne camoufle pas la logique du traitement.

Les données qui sont l'objet du traitement sont hiérarchisées en niveaux verticaux : fichiers, enregistrements, champs, caractères et bits par exemple. Une table peut être préparée pour chacun de ces niveaux. Cependant, il existe généralement aussi une organisation horizontale. Un fichier, ou une donnée quelconque, se trouve avoir une partie de début, un corps et une fin. On peut traduire dans l'organisation des tables cette division horizontale des données. A chaque niveau vertical peut correspondre une telle séparation horizontale, dès lors qu'un traitement demande une initialisation, une boucle et un traitement de sortie.

Pour être utile la division en tables doit encore réservée la nature des données et éviter de mélanger des données sans rapport entre elles. Au contraire une table peut être réservée à chaque niveau pour les vérifications de validité et les relations entre données.

Il peut arriver que les domaines n'aient pas d'organisation aussi structurée mais soient une succession d'unités, équivalentes à un enregistrement ou un message. C'est le cas des applications temps réel. A chaque message est attachée une priorité. L'heure d'arrivée et le contenu de chacun sont aléatoires. Dans de tels cas la séparation des tables doit tenir compte des priorités liées à la nature des messages dans chaque flot, et liées aux priorités hardware et aux priorités des traitements mis en cause.

Le plus important reproche fait aux tables de décisions est le manque de souplesse de leur emploi et la difficulté d'interprétation qu'elles offrent dans les programmes importants. L'utilisation de l'une ou l'autre des techniques de séparation présentées ici peut permettre de répondre à ces reproches.

Decision tables - Theory and Practice

S. L. Pollack, H. T. Hicks, W. J. Harrison

Ce livre a servi à l'élaboration du chapitre 2. Nous détaillerons ici l'étude théorique qu'il propose dans une annexe.

DEFINITION THEORIQUE DES TABLES DE DECISIONS

Conditions

Une condition est formée de deux opérandes et d'un opérateur. L'un au moins des opérandes est une variable. L'autre peut être une variable ou une constante. Les opérateurs possibles sont :

$$> \quad < \quad = \quad \geqslant \quad \leqslant$$

Une condition C_i peut être vraie, sa valeur $V(C_i)$ vaut alors 1, ou fausse et $V(C_i) = 0$. Nous noterons $a_i = V(C_i)$.

Deux conditions C_i et C_j sont indépendantes si elles n'ont aucun opérande en commun. Le couple a_i, a_j peut prendre toutes les valeurs :

$$(0,0) \quad (0,1) \quad (1,0) \text{ et } (1,1)$$

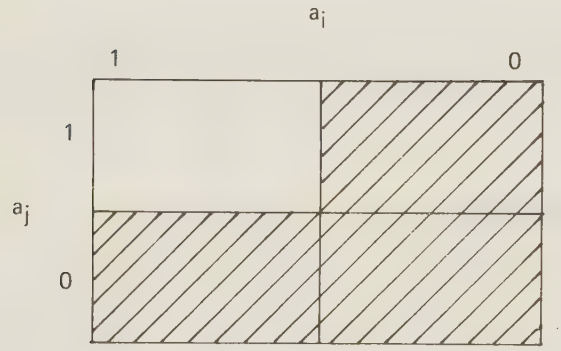
Deux conditions sont dites dépendantes si elles ont au moins un opérande en commun. On distingue deux sortes de dépendances :

- L'exclusion mutuelle, et dans ce cas a_i et a_j ne peuvent valoir 1 pour les mêmes valeurs des opérandes ;

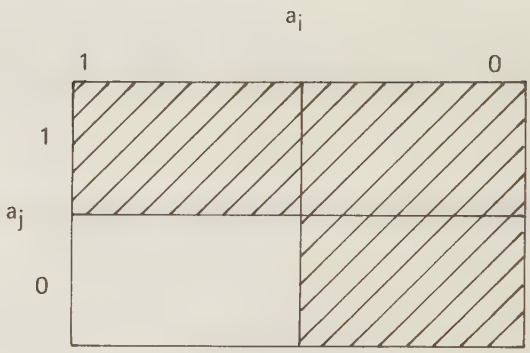
- la dépendance avec recouvrement, s'il existe un ensemble de valeurs pour les opérandes tel que les deux conditions sont satisfaites.

Le caractère de dépendance peut être reconnu sur la table de KARNAUGH

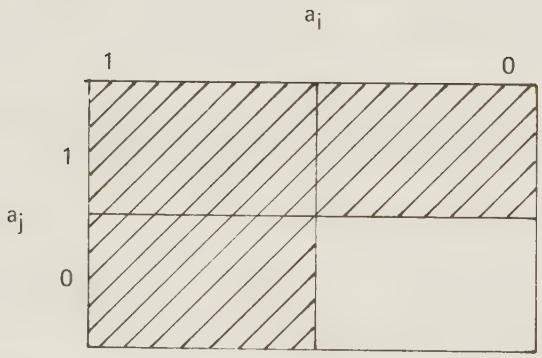
construites avec les valeurs des deux conditions :



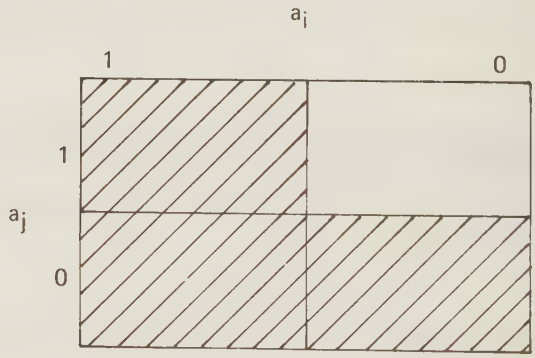
Exclusion mutuelle –



Recouvrement –



Recouvrement –



Recouvrement –

Entrées

Les entrées d'une règle j sur la condition i sont notées :

- Y_{ij} si la condition doit être vraie.
- N_{ij} si la condition doit être fausse.

- I_{ij} = Y_{ij} + N_{ij} si la condition i peut avoir l'une ou l'autre valeur.

- $*_{ij}$ la condition prend la valeur N si une condition dépendante prend la valeur explicite qui lui est demandée dans la règle j.

- $\$_{ij}$ la condition prend la valeur Y si une condition dépendante prend la valeur explicite qui lui est demandée dans la règle

et$n'ont pas à être vérifiée par la condition pour réaliser la règle. Cependant elles ont la pleine valeur de N et Y et dans ce sens sont des entrées implicites et non pas indifférentes.

Fonctions-ET d'une table

Soit W_{ij} une variable représentant une entrée : Y, N, I ou * $ et la fonction :

$$B_j = W_{1j} \cdot W_{2j} \cdot W_{3j} \cdot \quad \ldots \quad \cdot W_{nj}$$

B représente les entrées conditions de la règle j, Il y a 3^n fonctions B possibles, * et $ étant des cas particuliers de Y et N.

V (B_j) vaut 1 pour un jeu de valeur donné des opérandes des conditions si toutes les entrées sont vérifiées. Soit S = (a1, a2, ... , an) les valeurs des n conditions. La table suivante permet de construire B'_j :

W_{kj} =	et ak =	W'_{kj} vaut alors :
Y	0	0
Y	1	1
N	0	0
N	1	0
*	0	1
*	1	1
$	0	1
$	1	1
I	0 OU 1	1

et V (B_j) vaut 1 si tous les éléments de B'_j valent 1.

Dépendance des fonctions-ET d'une table

Deux fonctions-ET B_k et B_t d'une table sont dites dépendantes s'il existe un ensemble de valeur des conditions S = (a1, a2, ... , an) tel que les deux fonctions B_k et B_t sont vérifiées :

$$V(B_k) = V(B_t) = 1$$

Si un tel ensemble S n'existe pas les deux fonctions sont indépendantes. L'auteur annonce sur un exemple que si B_k et B_t sont indépendantes, elles diffèrent par au moins un couple Y, N relatif à la même condition.

Fonctions-OU d'une table

De la même façon, on peut définir des fonctions-OU inclusives à l'aide des entrées : Y, N, $\emptyset = \bar{I}$, *, $

Les résultats relatifs à ces fonctions se déduisent des résultats relatifs aux fonctions-ET par l'application des formules de De MORGAN :

$$\overline{(R + S)} = \bar{R} \cdot \bar{S}$$

Tables de décisions

Les tables de décisions classiques utilisent les fonctions d'entrées ET. Soit D_j une règle de décisions définie dans le langage courant par une phrase qui soumet l'exécution de certaines actions à la vérification de conditions.

a_{kj} note une action exécutée dans la règle D_j :

$$A_j = a_{1j} \odot a_{2j} \odot a_{3j} \odot \quad \cdots \quad \odot a_{mj}$$

résume toutes les actions de la règle. Le signe $\odot$ signifie que les actions sont exécutées en séquence.

D_j peut s'écrire $D_j = B_j \dashrightarrow A_j$ formule qui s'énonce : si $V(B_j) = 1$ les actions de A_j sont entreprises.

La table entière sera notée :

$$DT = D_1 \oplus D_2 \oplus D_3 \oplus \quad \ldots \quad \oplus D_q \; ; \; q \leqslant 3^n$$

Le symbole $\oplus$ qui lie deux règles a la signification du ou exclusif de l'algèbre de BOOL.

Fonctions propres - Règles simples

Une fonction-ET est dite propre si elle ne contient pas d'entrée I indifférente. Une règle $D_j = P_j \dashrightarrow A_j$ est dite simple si P_j est une fonction propre.

Une fonction-ET est dite mixte si elle contient une ou plusieurs entrées indifférentes. Une règle dont la fonction-ET est mixte, est dite complexe.

Les théorèmes relatifs aux fonctions-ET et OU d'une table sont déduits du seul axiome :

Dans une table, quelles que soient les valeurs des conditions une règle et une seule est satisfaite.

Théorèmes relatifs aux fonctions-ET d'une table

- Th. I

Dans une table, deux fonctions-ET sont indépendantes si et seulement si pour une position au moins, l'une des fonctions contient un Y ou $ et l'autre un N ou *. Elles sont dépendantes sinon.

Nous ne distinguerons pas pour la démonstration les entrées Y et $ d'une part * et N d'autre part. Soient B_r et B_s les deux fonctions-ET. B_r contient un Y à la position k : W_{kr} = Y de même par hypothèse W_{ks} = N. Les seuls jeux de valeurs S pour lesquels $V(B_r) = 1$ sont tels que a_k = Y. Pour ceux-là la position k de B_s' vaut zéro et $V(B_s) = 0$. B_r et B_s sont donc indépendantes. Réciproquement s'il n'existe pas un tel o couple d'entrées sur la même ligne, il est possible de construire une valeur de S qui vérifie les deux fonctions.

Celles-ci sont donc liées.

• Th. II

Dans une table, chaque fonction-ET propre est indépendante de toute autre fonction-ET propre.

1) Soit P_m une fonction propre ET :

$$P_m = Z_{1m} \cdot Z_{2m} \cdot Z_{3m} \quad \ldots \quad \cdot Z_{nm}$$

où Z vaut Y ou N. Il y a ainsi 2^n fonctions propres possibles pour un ensemble de n conditions données.

2) Une fonction-ET mixte qui contient une ou plusieurs entrées indifférentes peut être décomposée en fonctions propres liées par le ou exclusif. En effet I = Y + N. Il existe ainsi une sous-table de T :E formée de 2^n fonctions propres distinctes.

3) Considérons dans E tous les couples (P_r, P_s). P_r et P_s diffèrent par au moins une position : k pour laquelle $Z_{kr} = Y$ et $Z_{ks} = N$ ou le contraire. Les deux fonctions sont donc indépendantes.

Corollaire. Dans une table, il existe 2^n fonctions-ET propres. Chacune des $3^n - 2^n$ fonctions restant possibles sont mixtes. Les 2^n fonctions propres sont indépendantes.

• Th. III

Une fonction-ET mixte qui contient r entrées indifférentes peut être décomposée en une forme canonique de 2^r fonctions propres liées par ou exclusif.

La décomposition résulte des deux lois :

$$I = Y + N \text{ de définition}$$

$$\text{et } (R + S) \cdot T = R.T. + S.T \quad \text{de distributivité}$$

La fonction mixte M s'écrit alors $M = P_1 + P_2 + P_3 + P_4 \quad \ldots \quad + P_{2^r}$

• Th. IV

Dans une table T chaque fonction-ET mixte qui contient r entrées indifférentes est dépendante de chacune des 2^r fonctions pures de sa forme décomposée.

L'égalité de décomposition entraîne que si V (P_i) = 1 alors V (M) = 1 et P_i et M sont vérifiées pour le même ensemble S, donc dépendantes.

Une conséquence de ce théorème est que deux fonctions mixtes sont dépendantes si et seulement si elles ont une fonction propre commune dans leurs formes canoniques.

• Th. V

Une table, crée avec n conditions, contient un ensemble unique de 2^n fonctions-ET, indépendantes.

1) Il a déjà été montré qu'il existe au moins un tel ensemble soit C. Considérons tous les ensembles de 2^n fonctions formés à l'aide des 3^n fonctions possibles. Soit R un tel ensemble.

2) Soit t le nombre de fonctions mixtes que contient un élément de R. Nous limiterons R aux ensembles pour lesquels $1 \leqslant t \leqslant 2^n$. Pour C t vaut zéro et C n'est pas un élément de R.

3) Les éléments de R qui contiennent un ou plusieurs couples de fonctions mixtes qui se décomposent de façon à avoir deux à deux une fonction propre commune, ne contiennent pas 2^n fonctions indépendantes : c'est une conséquence du théorème IV.

4) Soit alors Q l'ensemble de tous les éléments de R qui ne contiennent pas deux à deux une fonction propre commune. Chaque fonction mixte d'un élément q de Q peut être décomposée en fonctions propres. La décomposition fait intervenir au moins 2 fonctions propres. Soit $t \geqslant 1$ le nombre de fonctions mixtes de q.

q contient $2^n - t + 2.t$ fonctions propres au moins soient $2^n + t$.

5) On sait qu'il y a 2^n fonctions propres différentes possibles dans la table. L'une au moins est donc répétée dans q.

6) Q contient tous les ensembles de t fonctions mixtes et $2^n - t$ fonctions propres comme $1 \leqslant t \leqslant 2^n$ il existe au moins 1 fonction propre dans chaque élément q de Q. Et une des fonctions propres de q est aussi atteinte par décomposition d'une des fonctions mixtes. Deux fonctions de q sont ainsi dépendantes. Et Q ne contient pas d'ensemble de 2^n fonctions indépendantes.

Le seul ensemble de 2^n fonctions indépendantes de la table est l'ensemble C des fonctions propres de la table.

• Th. VI

Une règle de décisions complexe dont la fonction-ET contient r entrées indifférentes est équivalente à 2^r règles simples.

Ce théorème résulte de la possibilité de décomposer une fonction-ET mixte en 2^r fonctions propres.

Théorèmes relatifs aux fonctions-OU d'une table

Conversion d'une fonction-OU en fonctions-ET.

Soit $\emptyset = \bar{I} = \overline{(Y+N)} = Y.N$ $\emptyset$ représente l'entrée nulle, qui ne peut être vérifiée. Une fonction-ET : $B_j = W_{1j} \cdot W_{2j} \cdot \quad \ldots \quad \cdot W_{nj}$ où W représente les entrées possibles Y N I * et $

de la même façon une fonction-OU sera notée :

$$E_j = U_{1j} + U_{2j} \quad \ldots \quad + U_{nj}$$

U représente l'une des entrées Y N $\emptyset$ * $. Il y a encore 3^n fonctions-OU possibles dans une même table. Le signe + représente l'opérateur binaire ou inclusif.

La conversion d'une fonction-OU en fonction-ET est créée par l'algorithme suivant :

a) Si $U_{1j} \neq \emptyset$ on crée $B_1 = U_{1j} \cdot I_2 \cdot I_3 \quad \ldots \quad \cdot I_n$

sinon on ne crée aucune fonction-ET, nous noterons ce fait par $B_1 = 0$

b) Si $U_{2j} \neq \emptyset$ on crée $B_2 = \bar{U}_{1j} \cdot U_{2j} \cdot I_3 \quad \ldots \quad \cdot I_n$

sinon $B_2 = 0$

.....

Enfin soit $G = B_1 + B_2 + B_3 \quad \ldots \quad + B_n$

où $B_i = 0$ si $U_{ij} = \emptyset$

et si $U_{ij} \neq \emptyset$

$$B_i = \overline{\overline{U_{1j}}} \cdot \overline{\overline{U_{2j}}} \cdot \overline{\overline{U_{3j}}} \quad \ldots \quad \overline{\overline{U_{i-1,j}}} \cdot U_{ij} \cdot I_{i+1,j} \cdot I_{i+2,j} \quad \ldots \quad \cdot I_n$$

On voit que la forme G créée est modifiée si l'ordre des conditions est modifié. Les B_i ne sont pas symétriques pour les U_{ij}. Cependant si l'on décompose chacun des B_i qui contiennent des entrées indifférentes, dans la forme canonique associée, l'expression de la forme G est symétrique et unique. E et G ont la même table de vérité, par construction et définissent donc la même structure logique.

• Th. VI'

Une fonction-OU qui possède r entrées nulles $\emptyset$ est équivalente à $2^n - 2^r$ fonctions-ET propres et distinctes.

Soit une fonction-OU dont les r dernières positions sont nulles :

$$E = Z_1 + Z_2 + Z_3 \quad \ldots \quad Z_{n-r} + \emptyset_{n-r+1} \quad \ldots \quad + \emptyset_n$$

E peut être traduit par :

$$G = (Z_1 \cdot I_2 \quad \ldots \quad I_n) \quad (\overline{\overline{Z_1}} \cdot Z_2 \, , Z_3 \quad \ldots \quad I_n) \quad (\quad \ldots \quad)$$

$$(\overline{\overline{Z_1}} \cdot \overline{\overline{Z_2}} \quad \ldots \quad \cdot \overline{\overline{Z_{n-r-1}}} \cdot Z_{n-r} \cdot I_{n-r+1} \quad \ldots \quad I_n)$$

Le premier terme contient n-1 entrées indifférentes et est équivalent à 2^{n-1} fonctions-ET propres. De même au second terme correspond 2^{n-2} fonctions-ET propres et au dernier 2^r. G est donc équivalent à $2^r + 2^{r+1} \quad \ldots \quad 2^{n-1}$ soit $2 - 2^r$ fonctions-ET propres d'où le théorème.

• Th. IV'

Il en résulte qu'une fonction-OU qui contient r entrées nulles est dépendante de $2^n - 2^r$ fonctions-ET propres.

Il suffit pour le montrer d'utiliser la décomposition mise en évidence par le théorème précédent.

• Th. I'

Deux fonctions-OU sont dépendantes si dans une position au moins elles ont toutes les deux un Y ou toutes les deux un N. Elles sont indépendantes sinon.

Supposons que Z_{k1} et Z_{k2} valent Y pour deux fonctions-OU E1 et E2. Un essai S pour lequel a_k vaut 1 impose que :

V (E1) = V (E2) = 1 et les fonctions sont dépendantes

Réciproquement soient E1 et E2 dont les positions précisées (i.e. Y ou N) sont toutes différentes. Les seuls couples qu'on puisse trouver sont :

Y	N
Ø	Y
Ø	N
Ø	Ø

Les positions pour lesquelles E1 ou E2 contiennent Ø interdisent que les deux fonctions soient vérifiées simultanément. Les autres positions sont alors Y pour l'une des fonctions, N pour l'autre ; on ne peut construire un essai S qui les satisfasse à la fois. Pour tout ensemble S une position de E_1' ou de E_2' sera nulle et V (E_1') ou V (E_2') sera nul. Les deux fonctions sont indépendantes.

• Th. I"

Une fonction-ET B et une fonction-OU E sont dépendantes si une position contient l'un des couples :

dans B	Y	et dans E	Y
ou	N		N
ou	I		Y
ou	I		N

Elles sont indépendantes sinon.

Soit $B = Y_1 \cdot W_2 \quad \ldots \quad W_n$ ou $B = I_1 \cdot W_2 \quad \ldots \quad \cdot W_n$

et $E = Y_1 \cdot U_2 \quad \ldots \quad U_n$

Pour S = (1, a2, a3, ... an) V (B) = V (E) = 1

B et E sont donc deux fonctions dépendantes.

A I - b 3. Recette des programmes à l'aide des tables de décisions

Ce sujet particulier est très peu traité dans la littérature. Un grand nombre d'articles l'aborde de façon très générale dans leurs conclusions, pour affirmer qu'une documentation et une analyse systématiques du problème posé sont une aide précieuse pour le contrôle et le débourrage d'un programme. L'étude résumée brièvement ici, cherche à construire de façon automatique des valeurs de test significatives, à partir d'une table de décisions. La difficulté provient de ce qu'un programme réel ne peut être traduit que par un grand nombre de tables classiques. Le problème posé par ce que nous avons appelé rupture sémantique et par les boucles du programme reste entier. L'étude résumée est signalée en référence 25.

Société d'Etude des Systèmes d'Automation

Contrat CRI-SESA N° 70021
Sept. 1971

Un générateur automatique de fichier de test

G. A. F. T.

Le but de GAFT est de créer automatiquement, à partir des documents d'analyse des jeux d'essais, de telle manière que le programme écrit à partir des mêmes documents soit intégralement testé. Le document choisi est une table de décisions.

I- Représentation des informations de la table de décisions

Les identificateurs de la partie condition sont les variables d'état, dont les valeurs sont responsables du choix de telle ou telle règle à l'exécution. Pour représenter les relations entre variables d'état de la souche condition, on associe à chaque identificateur le sommet d'un graphe.

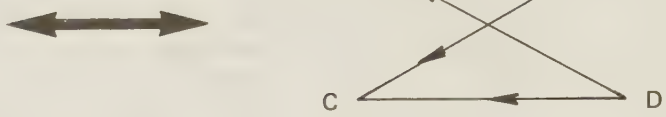
Les relations à représenter sont de trois types :

a) les relations d'ordre $<, \leqslant, >, \geqslant$

la relation $A < B$ se traduit par l'existence d'une arête orientée A--➤B joignant A à B. La relation $\leqslant$ se traduit de la même façon

mais, il faut, de plus, mémoriser le fait que les deux variables peuvent être égales. Exemple :

A < B
B < C
A > D
D < C



Ce type de relation d'ordre se traduit donc par l'existence d'un graphe orienté entre les différents sommets. On choisit de représenter ce graphe par un chaînage entre chaque élément et ses successeurs. Ce type de représentation n'étant possible que pour une structure arborescente, chaque fois qu'un élément a plusieurs antécédents, on crée un second élément identique au premier. Pour chaque identificateur l'information élémentaire sera formée de l'adresse du premier suivant et de l'adresse du prochain suivant de l'antécédent. La racine de l'arbre est pointée par une mémoire particulière.

b) La relation d'égalité

Les identificateurs de variables égales sont chaînés entre eux.

c) La relation d'inégalité

La relation A ≠ C se traduit par l'existence d'un chaîne entre A et C. Cette relation n'étant pas transitive, si on a de plus A ≠ B, on crée une variable égale à A, soit A' et on traduit A ≠ B par A' ≠ B qu'on stocke de la même façon.

En résumé les informations caractéristiques d'un identificateur sont les suivantes :

- adresse du premier suivant
- adresse du prochain suivant de l'antécédent
- adresse du premier identificateur égal
- adresse d'un identificateur lié avec celui-ci par la relation ≠
- booléen d'égalité acceptée avec le précédent.

II- Algorithme d'affectation

La construction d'un jeu de valeurs d'essais proviendra de l'affectation de valeurs aux variables d'état de la table. Pour chaque variable, l'utilisateur fournit l'intervalle de variation (bornes). Le long d'un chemin du graphe, les valeurs des variables associées aux identificateurs doivent être croissantes.

L'algorithme d'affectation pour un chemin du graphe est le suivant :

On attribue au premier identificateur la valeur de sa borne inférieure et au suivant la valeur de sa borne inférieure si elle est supérieure à la borne inférieure du précédent ; sinon, deux cas peuvent se produire :

- La relation entre les deux identificateurs est une relation $\leqslant$. Dans ce cas, on donne à l'identificateur la même valeur qu'à l'identificateur précédent si cette valeur est inférieure à la borne supérieure.

- La relation entre les deux identificateurs est une relation $<$. On attribue à l'identificateur la valeur de l'identificateur précédent + h si cette valeur est inférieure à la borne supérieure. (h étant fonction de l'intervalle minimal de variation.)

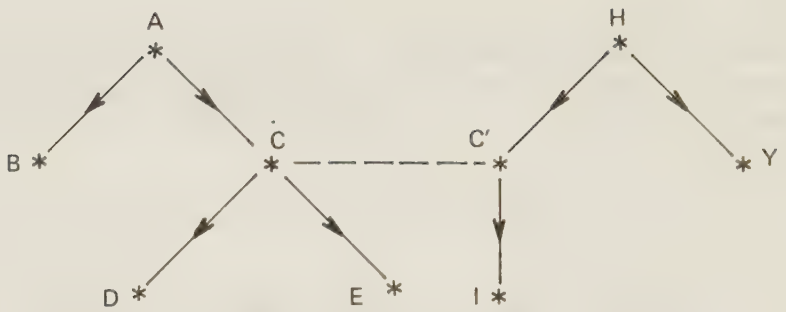
En fait, nous avons supposé que tous les identificateurs du chemin considéré n'étaient pas ceux de variables déjà affectées, ce qui peut se produire. En effet, deux éléments de deux chemins distincts peuvent être identiques, On a donc pu attribuer une valeur à cet élément lors d'un cheminement précédent.

Deux éventualités sont alors possibles :

- La valeur déjà affectée est supérieure à la valeur de l'élément précédent le long du chemin considéré et l'on conserve cette valeur.

- La valeur déjà affectée est inférieure et on l'augmente de la différence si c'est possible, mais il faut de plus modifier en conséquence les valeurs des suivants.

Exemple :



En résumé, le module d'affectation fonctionnera ainsi : on lui fournit la valeur associée à la variable du premier identificateur. Il attribue une valeur à chacun des suivants dans l'ordre des chemins du graphe. Si un de ces suivants a déjà une valeur attribuée et que l'on doive la modifier ainsi que celle des identificateurs égaux, on réappelle le module d'affectation pour celui des deux identificateurs de la chaîne des égaux qui ont des successeurs dont la valeur est déjà attribuée. Le module d'affectation est donc récursif à n niveaux, n étant la profondeur maximum autorisée par le récursivité.

Si les relations de la table de décisions sont impossibles à vérifier ; le programme sort en erreur après avoir atteint la profondeur maximum définie.

III- Utilisation

Le système GAFT est composée de deux phases distinctes qui ne sont pas présentes simultanément en mémoire centrale. Elles peuvent être appelée séparément. La première attribue des valeurs aux variables d'état de la table de décisions de telle sorte que l'ensemble des relations portant sur ces variables soient vérifiées. La seconde constitue des fichiers d'essais du programme à partir de renseignements sur leur structure et de la place des variables dans ces fichiers.

L'utilisateur aura le choix ou bien de les faire exécuter successivement ou bien de faire exécuter seulement le deuxième module de mise en forme des valeurs qu'il donnera lui-même.

Les données propres à la première phase sont :

- Les noms des variables d'état auxquelles le programme attribuera une valeur et leur format.

- Bornes inférieures et supérieures des variables d'état.

- La partie condition de la table de décisions (formes étendues).

ANNEXE II

EXEMPLE D'APPLICATION DE LA CHAINE TABOL

L'exemple dont les résultats d'exécution sont fournis ci-joint est introduit dans le chapitre 7 : UTILISATION DE LA CHAINE TABOL. On y trouvera les commentaires nécessaires à sa compréhension.

Ci-joint les différents listing d'exécution suivants :

A II - a. Les trois premières divisions du programme

A II - b. La procédure division

A II - c. Les tables de décisions à entrées limitées

A II - d. Le fichier des directives de transformation

A II - e. Un premier cycle de tables à entrées étendues.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    CONTROLE DES POINTAGES.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    CII-10070.
OBJECT-COMPUTER.
    CII-10070.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT JK-MAT ASSIGN TO
    MAT1
    ACCESS IS SEQUENTIAL
    ACTUAL KEY IS JK-ID.
    SELECT AC-CART ASSIGN TO
    CAR1.
    SELECT DB-CARTBON ASSIGN TO
    BON1
    ACCESS IS RANDOM
    ACTUAL KEY IS DB-CLE.
    SELECT FE-CARTFAU ASSIGN TO
    FAU1.
    SELECT ML-AFF ASSIGN TO
    AFF2
    ACCESS IS SEQUENTIAL.
    SELECT MINI-AFF  ASSIGN TO
    AFFM
    ACCESS IS SEQUENTIAL
    ACTUAL KEY IS MI-CLE.
    SELECT LM-AFF ASSIGN TO
    AFFL
    ACCESS IS RANDOM
    ACTUAL KEY IS LM-CLE.
 DATA DIVISION.
 FILE SECTION.
FD  ML-AFF
    BLOCK CONTAINS 1024
    LABEL RECORD STANDARD  DATA RECORD ML-ENR.
01  ML-ENR.
    02 ML-DLC   PICTURE X.
    02 ML-CLE.
        03 ML-AF PICTURE X(5).
        03 ML-SEC PICTURE XXX.
    02 ML-DEB   PICTURE X(4).
    02 ML-FIN   PICTURE X(4).
    02 ML-RESTE PICTURE X(67).
  FD  MINI-AFF
    BLOCK CONTAINS 1024
    LABEL RECORD STANDARD  DATA RECORD MINI-ENR.
01  MINI-ENR.
    02  MI-CLE  PICTURE X(8).
    02  MI-FIN  PICTURE X(4).
FD  AC-CART
```

```
      BLOCK CONTAINS 4096
    LABEL RECORD STANDARD  DATA RECORD AC-ENR.
01  AC-ENR.
    02 AC-SEC    PICTURE XXX.
    02 AC-ID     PICTURE XXXX.
    02 AC-SEM    PICTURE XX.
    02 AC-AFSE
       OCCURS 1 TO 10 TIMES.
      03 AC-AF   PICTURE X(5).
      03 AC-SE   PICTURE XX.
    02 FILLER    PICTURE X.
FD  JK-MAT
     BLOCK CONTAINS 1024
    LABEL RECORD STANDARD  DATA RECORD JK-ENR.
01  JK-ENR.
     02 JK-DLC   PICTURE X.
    02 JK-ID     PICTURE XXXX.
    02 JK-NOM    PICTURE X(20).
    02 JK-PREN   PICTURE X(15).
    02 JK-ITO    PICTURE X.
    02 JK-DEPT   PICTURE X.
    02 JK-SEC    PICTURE XXX.
 FD FE-CARTFAU
     BLOCK CONTAINS 4096
    LABEL RECORD STANDARD  DATA RECORD FE-ENR.
01  FE-ENR.
    02 FE-SEC    PICTURE XXX.
    02 FE-ID     PICTURE XXXX.
    02 FE-SEM    PICTURE 99.
    02 FE-RESTE.
      03 FE-AFSE
        OCCURS 1 TO 10.
       04 FE-AF PICTURE X(5).
       04 FE-SE PICTURE XX.
      03 FE-ITO PICTURE X.
    02 FE-ERIND PICTURE X(20).
    02 FE-ER1
     REDEFINES FE-ERIND.
       03 FE-ERR
         OCCURS 10  PICTURE 9.
       03 FE-IND
         OCCURS 10 PICTURE 9.
FD  DB-CARTBON
     BLOCK CONTAINS 1024
    LABEL RECORD IS STANDARD  DATA RECORD DB-ENR.
01  DB-ENR.
    02 DB-SEC    PICTURE XXX.
    02 DB-CLE.
      03 DB-SEM PICTURE 99.
      03 DB-ID  PICTURE XXXX.
     .
    02 DB-RESTE.
      03 DB-AFSE
         OCCURS 1 TO 10.
```

```
        04 DB-AF PICTURE X(5).
        04 DB-SE PICTURE XX.
FD  LM-AFF
     BLOCK CONTAINS 1024
    LABEL RECORD STANDARD  DATA RECORD LM-ENR.
01  LM-ENR.
    02 LM-CLE.
      03 LM-AF  PICTURE X(5).
      03 LM-SEC  PICTURE 999.
    02  LM-FIN  PICTURE X(4).
 WORKING-STORAGE SECTION.
77  W71-SEM      PICTURE 99.
77  W72-SEC      PICTURE 999.
77  W73-A        PICTURE 9  VALUE 1.
77  W74-AF       PICTURE 9(5).
77  W75-CONTSE   PICTURE 99 VALUE 0.
77  AIG1         PICTURE 9  VALUE 0.
77  AIG2         PICTURE 9  VALUE 0.
77  AIG3         PICTURE 9  VALUE 0.
77  AIG4         PICTURE 9  VALUE 0.
77  AIG5         PICTURE 9  VALUE 0.
77  AIG6         PICTURE 9  VALUE 0.
77  AIG7         PICTURE 9  VALUE 0.
77  AIG8         PICTURE 9  VALUE 0.
77  AIG9         PICTURE 9  VALUE 0.
77  AIG10        PICTURE 9  VALUE 0.
77  AIG11        PICTURE 9  VALUE 0.
77  AIG12        PICTURE 9  VALUE 0.
 77 AIGEND PICTURE 9 VALUE 0.
77  I            PICTURE 99  VALUE 1.
77  J            PICTURE 99  VALUE 10.
01  W1-ZONAC.
    02 W1-ZONA.
     03 W1-SEC  PICTURE 999.
     03 W1-ID   PICTURE X(4).
     03 W1-SEM  PICTURE 99.
     03 W1-ZONB.
     04 W1-AFSE
       OCCURS 1 TO 10.
       05 W1-AF PICTURE X(5).
       05 W1-SE PICTURE 99.
     03 W1-ITO  PICTURE X.
    02 W1-ERR.
      03 W1-ERA PICTURE 9(4).
      03 W1-ERS PICTURE 9.
      03 W1-ERB PICTURE 99.
      03 W1-ERC PICTURE 9.
      03 W1-ERD PICTURE 99.
      03 W1-INDO.
        04 W1-IND   OCCURS 10  PICTURE 9.
     02 W11-ERR
    REDEFINES W1-ERR.
    03 W11-ER
    OCCURS 20   PICTURE 9.
```

```
 3    INIT-MINI.
 5        OPEN  INPUT ML-AFF
               OUTPUT  MINI-AFF
           .
      FAB-MINI.
 8              READ ML-AFF AT END GO TO INIT.
10        IF ML-SEC = '000'
12        MOVE ML-CLE TO MI-CLE
          MOVE ML-FIN  TO MI-FIN
13        WRITE MINI-ENR  INVALID KEY
16        DISPLAY 'IK MINI'  EXHIBIT NAMED ML-ENR
17        GO TO FAB-MINI.
18        GO TO FAB-MINI.
19   INIT.
22        CLOSE ML-AFF  MINI-AFF
            MOVE ZERO TO W1-ZONAC.
23        OPEN INPUT
24        AC-CART
          JK-MAT
          OPEN INPUT-OUTPUT
25        LM-AFF
          DB-CARTBON
          OPEN OUTPUT
26        FE-CARTFAU
          READ AC-CART AT END DISPLAY 'AC-CART VIDE'
29        STOP RUN.
30         MOVE AC-SEM TO W71-SEM.
31   DEB.
34        IF AIG1 = 0 GO TO 1B00.
35        MOVE 1 TO W11-ER(3).
36        GO TO 1B10.
37   1B00.
40        IF AIG2 = 1 GO TO 1A00.
41   1B05.
44        IF AIG3 = 1 GO TO 1B10.
45   LECABC.
48        READ JK-MAT AT END
50        MOVE 1 TO AIG2 GO TO 1A00.
51        IF JK-ID = 'AAAA' GO TO LECABC.
53        IF JK-ID = 'BBBB' GO TO LECABC.
55        IF JK-ID = 'CCCC' GO TO LECABC.
57   1B10.
60        IF JK-ID GREATER THAN AC-ID GO TO 1A00.
61        IF JK-ID LESS THAN AC-ID GO TO 1C00.
63        MOVE 0 TO AIG3.
64   1B15.
67        MOVE AC-ENR TO W1-ZONA
            .
      1B16.
70        READ AC-CART AT END
72         MOVE 1 TO AIGEND
           MOVE 0 TO AC-ID.
73        IF AC-ID = W1-ID GO TO 1C10.
75        MOVE 0 TO AIG1.
```

```
 76    1B20.
 79        MOVE JK-ITO TO W1-ITO.
           IF W1-SEC = JK-SEC GO TO 1B25.
 81        IF W11-ER(2) = 1 GO TO 1B25.
 83          IF W11-ER(3) = 1 MOVE 1 TO W11-ER(4) GO TO 1B25.
 86        MOVE 1 TO W11-ER(4), W11-ER(1).
 88    1B25.
 91        IF W1-SEM = W71-SEM GO TO 2B00.
 92        MOVE 1 TO W11-ER(5)
 93        MOVE W71-SEM TO W1-SEM.
 94    2B00.
 97         IF W1-AF(I) = SPACE GO TO 2A00.
 98        MOVE W1-AF(I) TO LM-AF
 99        MOVE ZERO TO LM-SEC.
100        ADD 10, I       GIVING J
101        READ LM-AFF INVALID KEY GO TO 2B05.
104        IF LM-FIN = SPACE GO TO 2B10.
106        MOVE 1 TO W11-ER(8)
107        MOVE 2 TO W11-ER(J) GO TO 2B10.
109    2B05.
112        MOVE 1 TO W11-ER(7), W11-ER(J)
113        IF W11-ER(3) = 1 GO TO 2B10.
115        MOVE 1 TO W11-ER(1).
116    2B10.
119         IF W1-SE(I) NOT NUMERIC MOVE 99 TO W1-SE(I)
120              .
            ADD W1-SE(I)  TO W75-CONTSE
121        ADD 1 TO I
122        IF I GREATER THAN 10 MOVE 1 TO I GO TO 2A00.
125        GO TO 2B00.
126    1A00.
129        MOVE 1 TO W11-ER(2)
           MOVE 1 TO AIG3
130        GO TO 1B15.
131    1A10.
134        IF AIG2 = 0 GO TO 1A15.
135        GO TO FIN.
136    1A15.
139        READ JK-MAT AT END
141        GO TO FIN.
            PERFORM MATMANQ THRU 3A20   GO TO 1A15.
143    1C00.
146        MOVE ZERO TO AIG3.
           PERFORM MATMANQ  THRU 3A20
147         MOVE ZERO TO W1-ERR
148        GO TO 1B05.
149    1C10.
152        IF AC-ENR = W1-ZONA GO TO 1B16.
153        MOVE 1 TO W11-ER(9), W11-ER(3)
155         IF W11-ER(2) = 1  MOVE 1 TO AIG1.
157        IF AIG1 = 0
159        MOVE 1 TO W11-ER(1)  AIG1.
160        GO TO 1B20.
161    2A00.
```

```
164         IF W75-CONTSE = 10 GO TO 2A05.
165         MOVE 1 TO W11-ER(6)
166         IF W11-ER(3) = 0  MOVE 1 TO W11-ER(1).
168     2A05.
171         IF W1-ERR = 0 GO TO 2A10.
172          IF W11-ER(2) = 1  MOVE 0 TO W11-ER(1)  GO TO 2A07.
175         IF W11-ER(1) = 1 GO TO 2A06.
177         IF W11-ER(9) = 1 GO TO 2A07.
179         MOVE 0 TO AIG4
180         GO TO 2A08.
181     2A06.
184         IF W11-ER(4) = 1  OR W11-ER(3) = 1
186         MOVE ZERO TO W11-ER(1)
            MOVE 1 TO W11-ER(10)
187          PERFORM ECRIFAU
188          MOVE JK-ITO TO DB-ITO
189         MOVE JK-SEC TO DB-SEC FE-SEC
191         MOVE JK-ID  TO DB-ID  FE-ID
193         MOVE W71-SEM TO DB-SEM  FE-SEM
195          MOVE SPACE TO DB-RESTE,W1-ZONB
196         PERFORM 3A15 THRU 3A20  GO TO 2A09
198                 .
            MOVE 1 TO AIG4  GO TO 2A08.
200     2A07.
203          PERFORM ECRIFAU.
              GO TO 2A20.
204     2A08.
207          PERFORM ECRIFAU
            IF AIG4 = 1 MOVE SPACE TO W1-ZONB.
209         MOVE W1-ZONA TO DB-ENR
210         MOVE W1-ID TO DB-ID
211         MOVE W71-SEM TO DB-SEM.
212     2A09.
215         WRITE DB-ENR INVALID KEY
217          GO TO 2A091.
            GO TO 2A20
218           .
        2A091.
221          MOVE W1-SEM TO DB-SEM   MOVE W1-ID  TO  DB-ID
222          PERFORM  2A15 THRU  1. GO TO 2A20.
224         DISPLAY 'IK 2A09 MATRICULE DEJA ECRIT'  GO TO 2A20.
226     2A10.
229         MOVE W1-ZONA TO DB-ENR
            MOVE W1-ID  TO DB-ID
230         MOVE W1-SEM TO DB-SEM
231         WRITE DB-ENR INVALID KEY
234         MOVE W1-SEM TO DB-SEM
            MOVE W1-ID  TO DB-ID GO TO 2A15. GO TO 2A20.
237     2A15.
240         READ DB-CARTBON INVALID KEY GO TO FIN.
242     1.
245         IF DB-RESTE = SPACE
246          MOVE W1-ZONB TO DB-RESTE
             REWRITE DB-ENR  INVALID KEY GO TO 1.
```

```
249     2A20.
252         MOVE 1 TO I
            MOVE 0 TO W75-CONTSE, AIG4, W1-ERR, FE-ERIND
256          MOVE ZERO TO W1-ERR
257         MOVE SPACE TO W1-ZONA
258          IF AIGEND = 1 GO TO 1A10.
260        GO TO DEB.
261     FIN.
264         STOP RUN.
        MATMANQ.
267         MOVE W71-SEM TO DB-SEM
            MOVE JK-ID   TO DB-ID
268         MOVE JK-SEC  TO DB-SEC
269          MOVE JK-ITO TO DB-ITO
270         READ DB-CARTBON INVALID KEY
273         GO TO 3A05.
             GO TO 3A07.
274     3A05.
277         MOVE SPACE TO DB-RESTE
            WRITE DB-ENR INVALID KEY GO TO 3A10.
280     3A07.
283         IF DB-RESTE = SPACE GO TO 3A10.
284         GO TO 3A20.
285     3A10.
288         MOVE W71-SEM TO FE-SEM
            MOVE JK-ID TO FE-ID
289         MOVE JK-SEC TO FE-SEC
290           .
        3A15.
293          MOVE ZERO TO FE-ERIND.
            MOVE 1 TO FE-ERR(1)
294         MOVE SPACE TO FE-RESTE
295         WRITE FE-ENR .
296     3A20.
299         EXIT.
        ECRIFAU.
302         WRITE FE-ENR FROM WI-ZONAC
303           .
303      INSTRUCTIONS DANS LE PROGRAMME
```

```
                              * TABLE OUVERTE INIT-MINI *
                              ****************************

  TABLE NUMERO   1

   REGLES   1   A     1
  ------------------------------------------II---I
                                            II   I
1 OPEN      INPUT ML-AFF OUTPUT MINI-AFF    II * I
                                            II   I
  ------------------------------------------II---I
                                  RUPTURES DE SEQUENCE
                                            II   I
2 GO        TO FAB-MINI ..................  II * I
                                            II   I
  ------------------------------------------II---I
```

```
                                * TABLE OUVERTE FAB-MINI *
                                **************************

  TABLE NUMERO   2

   REGLES   1   A     4
  ------------------------------------II---I---I---I---I
                                      II   I   I   I   I
1  ONE MORE RECORD IN ML-AFF ........ II O I O I O I N I
                                      II   I   I   I   I
  ------------------------------------II---I---I---I---I
                                SOUS-TABLE DE DEGRE   1
                                      II   I   I   I   I
2  ML-SEC = '000' ................... II O I O I N I   I
                                      II   I   I   I   I
  ------------------------------------II---I---I---I---I
                                SOUS-TABLE DE DEGRE   2
                                      II   I   I   I   I
3  MI-CLE VALID KEY ................. II O I N I   I   I
                                      II   I   I   I   I
  ------------------------------------II---I---I---I---I
  ------------------------------------II---I---I---I---I
4 READ     ML-AFF RECORD ............ II * I * I * I   I
                                      II   I   I   I   I
  ------------------------------------II---I---I---I---I
                                SOUS-TABLE DE DEGRE   1
                                      II   I   I   I   I
5 MOVE     ML-CLE TO MI-CLE ......... II * I * I   I   I
                                      II   I   I   I   I
6 MOVE     ML-FIN TO MI-FIN ......... II * I * I   I   I
                                      II   I   I   I   I
  ------------------------------------II---I---I---I---I
                                SOUS-TABLE DE DEGRE   2
                                      II   I   I   I   I
7 WRITE    MINI-ENR ................. II * I   I   I   I
                                      II   I   I   I   I
8 DISPLAY  'IK MINI' ................ II   I * I   I   I
                                      II   I   I   I   I
9 EXHIBIT  NAMED ML-ENR .............. II   I * I   I   I
                                      II   I   I   I   I
  ------------------------------------II---I---I---I---I
                                RUPTURES DE SEQUENCE
                                      II   I   I   I   I
10 GO      TO FAB-MINI .............. II * I * I * I   I
                                      II   I   I   I   I
11 GO      TO INIT .................. II   I   I   I * I
                                      II   I   I   I   I
  ------------------------------------II---I---I---I---I
```

```
                                * TABLE OUVERTE INIT *
                                **********************

   TABLE NUMERO   3

    REGLES   1   A     2
   ----------------------------------------------II---I---I
                                                 II   I   I
 1  ONE MORE RECORD IN AC-CART ..... II O I N I
                                                 II   I   I
   ----------------------------------------------II---I---I
   ----------------------------------------------II---I---I
 2 CLOSE     ML-AFF MINI-AFF ..... II * I * I
                                                 II   I   I
 3 MOVE      ZERO TO W1-ZONAC ..... II * I * I
                                                 II   I   I
 4 OPEN      INPUT AC-CART JK-MAT ..... II * I * I
                                                 II   I   I
 5 OPEN      INPUT-OUTPUT LM-AFF DB-CARTBON  II * I * I
                                                 II   I   I
 6 OPEN      OUTPUT FE-CARTFAU ..... II * I * I
                                                 II   I   I
 7 READ      AC-CART RECORD ..... II * I   I
                                                 II   I   I
 8 MOVE      AC-SEM TO W71-SEM ..... II * I   I
                                                 II   I   I
 9 DISPLAY   'AC-CART VIDE' ..... II   I * I
                                                 II   I   I
   ----------------------------------------------II---I---I
                                    RUPTURES DE SEQUENCE
                                                 II   I   I
10 GO        TO DEB ..... II * I   I
                                                 II   I   I
11 STOP      RUN ..... II   I * I
                                                 II   I   I
   ----------------------------------------------II---I---I
```

* TABLE OUVERTE DEB *
****

TABLE NUMERO 4

REGLES 1 A 5

1	AIG1 = 0	O	O	N	N	N			
2	AIG2 = 1	O	N						
3	JK-ID GREATER THAN AC-ID			O	N	N			
4	JK-ID LESS THAN AC-ID				O	N			
5	MOVE 1 TO W11-ER (3)				*	*	*		
6	MOVE 1 TO W11-ER (2)	*		*					
7	MOVE 1 TO AIG3	*		*					
8	MOVE ZERO TO AIG3				*				
9	PERFORM MATMANQ THRU 3A20				*				
10	MOVE ZERO TO W1-ERR				*				
11	MOVE 0 TO AIG3					*			
12	MOVE AC-ENR TO W1-ZONA	*		*		*			
	RUPTURES DE SEQUENCE								
13	GO TO 1816	*		*		*			
14	GO TO 1805		*		*				

```
                              * TABLE OUVERTE 1B05 *
                              **********************

  TABLE NUMERO   5

   REGLES   1   A     4
   ------------------------------------II---I---I---I---I---I
                                       II   I   I   I   I   I
1   AIG3 = 1 ..........................II O I O I O I N I
                                       II   I   I   I   I   I
2   JK-ID GREATER THAN AC-ID ..........II O I N I N I   I
                                       II   I   I   I   I   I
3   JK-ID LESS THAN AC-ID .............II   I O I N I   I
                                       II   I   I   I   I   I
   ------------------------------------II---I---I---I---I---I
   ------------------------------------II---I---I---I---I---I
4 MOVE      1 TO W11-ER (2) ..........II * I   I   I   I   I
                                       II   I   I   I   I   I
5 MOVE      1 TO AIG3 ................II * I   I   I   I   I
                                       II   I   I   I   I   I
6 MOVE      ZERO TO AIG3 .............II   I * I   I   I   I
                                       II   I   I   I   I   I
7 PERFORM   MATMANQ THRU 3A20 ........II   I * I   I   I   I
                                       II   I   I   I   I   I
8 MOVE      ZERO TO W1-ERR ...........II   I * I   I   I   I
                                       II   I   I   I   I   I
9 MOVE      0 TO AIG3 ................II   I   I * I   I   I
                                       II   I   I   I   I   I
10 MOVE     AC-ENR TO W1-ZONA ........II * I   I * I   I   I
                                       II   I   I   I   I   I
   ------------------------------------II---I---I---I---I---I
                                  RUPTURES DE SEQUENCE
                                       II   I   I   I   I   I
11 GO       TO 1B16 ..................II * I   I * I   I   I
                                       II   I   I   I   I   I
12 GO       TO 1B05 ..................II   I * I   I   I   I
                                       II   I   I   I   I   I
13 GO       TO LECABC ................II   I   I   I * I   I
                                       II   I   I   I   I   I
   ------------------------------------II---I---I---I---I---I
```

```
                                * TABLE OUVERTE LECABC *
                                ************************

   TABLE NUMERO   6

    REGLES   1   A     7
   ------------------------------------------II---I---I---I---I---I---I---I---I
                                              II   I   I   I   I   I   I   I   I
 1  ONE MORE RECORD IN JK-MAT ..... II O I O I O I O I O I O I N I
                                              II   I   I   I   I   I   I   I   I
   ------------------------------------------II---I---I---I---I---I---I---I---I
                                    SOUS-TABLE DE DEGRE   1
                                              II   I   I   I   I   I   I   I   I
 2  JK-ID = 'AAAA' ..................... II O I N I N I N I N I N I   I
                                              II   I   I   I   I   I   I   I   I
 3  JK-ID = 'BBBB' ..................... II   I O I N I N I N I N I   I
                                              II   I   I   I   I   I   I   I   I
 4  JK-ID = 'CCCC' ..................... II   I   I O I N I N I N I   I
                                              II   I   I   I   I   I   I   I   I
 5  JK-ID GREATER THAN AC-ID ......... II   I   I   I O I N I N I   I
                                              II   I   I   I   I   I   I   I   I
 6  JK-ID LESS THAN AC-ID ............ II   I   I   I   I O I N I   I
                                              II   I   I   I   I   I   I   I   I
   ------------------------------------------II---I---I---I---I---I---I---I---I
   ------------------------------------------II---I---I---I---I---I---I---I---I
 7 READ      JK-MAT RECORD ............ II * I * I * I * I * I * I   I
                                              II   I   I   I   I   I   I   I   I
 8 MOVE      1 TO AIG2 ................ II   I   I   I   I   I   I * I
                                              II   I   I   I   I   I   I   I   I
 9 MOVE      1 TO W11-ER (2) .......... II   I   I   I   I   I   I * I
                                              II   I   I   I   I   I   I   I   I
10 MOVE      1 TO AIG3 ................ II   I   I   I   I   I   I * I
                                              II   I   I   I   I   I   I   I   I
11 MOVE      AC-ENR TO W1-ZONA ........ II   I   I   I   I   I   I * I
                                              II   I   I   I   I   I   I   I   I
   ------------------------------------------II---I---I---I---I---I---I---I---I
                                    SOUS-TABLE DE DEGRE   1
                                              II   I   I   I   I   I   I   I   I
12 MOVE      1 TO W11-ER (2) .......... II   I   I   I * I   I   I   I   I
                                              II   I   I   I   I   I   I   I   I
13 MOVE      1 TO AIG3 ................ II   I   I   I * I   I   I   I   I
                                              II   I   I   I   I   I   I   I   I
14 MOVE      ZERO TO AIG3 ............. II   I   I   I   I * I   I   I   I
                                              II   I   I   I   I   I   I   I   I
15 PERFORM   MATMANQ THRU 3A20 ........ II   I   I   I   I * I   I   I   I
                                              II   I   I   I   I   I   I   I   I
16 MOVE      ZERO TO W1-ERR ........... II   I   I   I   I * I   I   I   I
                                              II   I   I   I   I   I   I   I   I
17 MOVE      0 TO AIG3 ................ II   I   I   I   I   I * I   I   I
                                              II   I   I   I   I   I   I   I   I
18 MOVE      AC-ENR TO W1-ZONA ........ II   I   I   I * I   I * I   I   I
                                              II   I   I   I   I   I   I   I   I
   ------------------------------------------II---I---I---I---I---I---I---I---I
                                    RUPTURES DE SEQUENCE
                                              II   I   I   I   I   I   I   I   I
19 GO        TO LECABC ................ II * I * I * I   I   I   I   I   I   I
                                              II   I   I   I   I   I   I   I   I
20 GO        TO 1B16 .................. II   I   I   I * I   I * I * I * I
                                              II   I   I   I   I   I   I   I   I
21 GO        TO 1B05 .................. II   I   I   I   I * I   I   I   I   I
                                              II   I   I   I   I   I   I   I   I
   ------------------------------------------II---I---I---I---I---I---I---I---I
```

TABLE NUMERO 7

REGLES 1 A 19	ES 20 A 41
1 ONE MORE RECORD IN AC-CART	O I N
SOUS-TABLE DE DEGRE 1	SOUS-TABLE DE DEGRE 1
2 AC-ID = W1-ID	O I N
3 AC-ENR = W1-ZONA	O I N
4 W1-SEC = JK-SEC	O I N
5 W11-ER (2) = 1	O I N
6 W11-ER (3) = 1	O I N
SOUS-TABLE DE DEGRE 2	SOUS-TABLE DE DEGRE 2
7 W11-ER (2) = 1	O I N
8 AIG1 = 0	O I N
9 W1-SEC = JK-SEC	O I N
10 W11-ER (3) = 1	O I N
SOUS-TABLE DE DEGRE 3	SOUS-TABLE DE DEGRE 3
11 AIG1 = 0	O I N
12 W1-SEC = JK-SEC	O I N
13 W11-ER (2) = 1	O I N
14 W11-ER (3) = 1	O I N
SOUS-TABLE DE DEGRE 4	SOUS-TABLE DE DEGRE 4
15 W11-ER (2) = 1	O I N
16 W11-ER (3) = 1	O I N
17 READ AC-CART RECORD	*
18 MOVE 1 TO AIGEND	*
19 MOVE 0 TO AC-ID	*
SOUS-TABLE DE DEGRE 1	SOUS-TABLE DE DEGRE 1
20 MOVE 1 TO W11-ER (9)	*
21 MOVE 1 TO W11-ER (3)	*
22 MOVE 0 TO AIG1	*
23 MOVE JK-ITO TO W1-ITO	*
24 MOVE 1 TO W11-ER (4)	*
25 MOVE 1 TO W11-ER (1)	*
SOUS-TABLE DE DEGRE 2	SOUS-TABLE DE DEGRE 2
26 MOVE 1 TO W11-ER (1)	*
27 MOVE 1 TO AIG1	*
28 MOVE JK-ITO TO W1-ITO	*
29 MOVE 1 TO W11-ER (4)	*
30 MOVE 1 TO W11-ER (1)	*
SOUS-TABLE DE DEGRE 3	SOUS-TABLE DE DEGRE 3
31 MOVE 1 TO W11-ER (1)	*
32 MOVE 1 TO AIG1	*
33 MOVE JK-ITO TO W1-ITO	*
34 MOVE 1 TO W11-ER (4)	*
35 MOVE 1 TO W11-ER (1)	*
SOUS-TABLE DE DEGRE 4	SOUS-TABLE DE DEGRE 4
36 MOVE 1 TO W11-ER (4)	*
37 MOVE 1 TO W11-ER (1)	*
RUPTURES DE SEQUENCE	RUPTURES DE SEQUENCE
38 GO TO 1816	*
39 GO TO 1825	*

```
                                  * TABLE OUVERTE 1825 *
                                  **********************

  TABLE NUMERO   8

   REGLES   1   A     2
   ------------------------------II---I---I
                                  II   I   I
1   W1-SEM = W71-SEM ..........   II O I N I
                                  II   I   I
   ------------------------------II---I---I
   ------------------------------II---I---I
2 MOVE      1 TO W11-ER (5) ..    II   I * I
                                  II   I   I
3 MOVE      W71-SEM TO W1-SEM     II   I * I
                                  II   I   I
   ------------------------------II---I---I
                                  RUPTURES DE SEQUENCE
                                  II   I   I
4 GO        TO 2B00 ...........   II * I * I
                                  II   I   I
   ------------------------------II---I---I
```

* TABLE OUVERTE 2800 *
****

TABLE NUMERO 9

REGLES 1 A 17

1 W1-AF (I) = SPACE	O	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SOUS-TABLE DE DEGRE 1																	
2 LM-CLE VALID KEY		O	O	O	O	O	O	O	O	N	N	N	N	N	N	N	N
SOUS-TABLE DE DEGRE 2																	
3 LM-FIN = SPACE		O	O	O	O	N	N	N	N								
4 W1-SE (I) NUMERIC		O	O	N	N	O	O	N	N	O	O	N	N	O	O	N	N
5 W11-ER (3) = 1										O	O	O	O	N	N	N	N
SOUS-TABLE DE DEGRE 3																	
6 I GREATER THAN 10		O	N	O	N	O	N	O	N	O	N	O	N	O	N	O	N
7 MOVE W1-AF (I) TO LM-AF		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
8 MOVE ZERO TO LM-SEC		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
9 COMPUTE J = 10 + I		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
SOUS-TABLE DE DEGRE 1																	
10 READ LM-AFF RECORD		*	*	*	*	*	*	*	*								
11 MOVE 1 TO W11-ER (7)										*	*	*	*	*	*	*	*
12 MOVE 1 TO W11-ER (J)										*	*	*	*	*	*	*	*
SOUS-TABLE DE DEGRE 2																	
13 MOVE 1 TO W11-ER (8)						*	*	*	*								
14 MOVE 2 TO W11-ER (J)						*	*	*	*								
15 MOVE 1 TO W11-ER (1)														*	*	*	*
16 MOVE 99 TO W1-SE (I)				*	*			*	*			*	*			*	*
17 COMPUTE W75-CONTSE = W1-SE (I) + W75-CONTSE		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
18 COMPUTE I = 1 + I		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
SOUS-TABLE DE DEGRE 3																	
19 MOVE 1 TO I		*		*		*		*		*		*		*		*	
RUPTURES DE SEQUENCE																	
20 GO TO 2A00	*	*		*		*		*		*		*		*		*	
21 GO TO 2B00			*		*		*		*		*		*		*		*

```
                                  * TABLE OUVERTE 1A15 *
                                  **********************

  TABLE NUMERO  10

   REGLES   1   A     2
  ------------------------------------II---I---I
                                      II   I   I
1  ONE MORE RECORD IN JK-MAT ........ II O I N I
                                      II   I   I
  ------------------------------------II---I---I
  ------------------------------------II---I---I
2 READ     JK-MAT RECORD ............ II * I   I
                                      II   I   I
3 PERFORM  MATMANQ THRU 3A20 ........ II * I   I
                                      II   I   I
  ------------------------------------II---I---I
                                RUPTURES DE SEQUENCE
                                      II   I   I
4 GO       TO 1A15 ................. II * I   I
                                      II   I   I
5 STOP     RUN ..................... II   I * I
                                      II   I   I
  ------------------------------------II---I---I
```

* TABLE OUVERTE 2A00 *

TABLE NUMERO 11

REGLES 1 A 19 … ES 20 A 27

1	W75-CONTSE = 10	O	O	O	O	O	O	O	O	O	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	
2	W1-ERR = 0	O	N	N	N	N	N	N	N	N																			
3	W11-ER (2) = 1		O	N	N	N	N	N	N	N																			
4	W11-ER (1) = 1			O	O	O	O	N	N	N																			
5	W11-ER (3) = 1			O	N	N	N																						
6	W11-ER (4) = 1				O	N	N																						
7	W11-ER (9) = 1							O	N	N																			
	SOUS-TABLE DE DEGRE 1																												SOUS-TABLE DE DEGRE 1
8	AIG4 = 1					O	N		O	N																			
9	W11-ER (3) = 0										O	O	O	O	O	O	O	O	O	N	N	N	N	N	N	N	N	N	
10	W1-ERR = 0										O	N	N	N	N	N	N	N	N	O	N	N	N	N	N	N	N	N	
11	W11-ER (2) = 1																				O	N	N	N	N	N	N	N	
12	W11-ER (1) = 1																					O	O	O	O	N	N	N	
13	W11-ER (3) = 1																					O	N	N	N				
14	W11-ER (4) = 1																						O	N	N				
15	W11-ER (9) = 1																									O	N	N	
	SOUS-TABLE DE DEGRE 2																												SOUS-TABLE DE DEGRE 2
16	W11-ER (2) = 1											O	N	N	N	N	N	N	N										
17	W11-ER (1) = 1												O	O	O	O	N	N	N										
18	W11-ER (3) = 1												O	N	N	N													
19	W11-ER (4) = 1													O	N	N													
20	W11-ER (9) = 1																O	N	N										
21	AIG4 = 1																							O	N		O	N	
	SOUS-TABLE DE DEGRE 3																												SOUS-TABLE DE DEGRE 3
22	AIG4 = 1														O	N		O	N										
23	MOVE 0 TO W11-ER (1)		*																										
24	MOVE ZERO TG W11-ER (1)			*	*																								
25	MOVE 1 TO W11-ER (10)			*	*																								
26	MOVE 1 TO AIG4					*	*																						
27	MOVE 0 TO AIG4								*	*																			
28	PERFORM ECRIFAU		*	*	*	*	*	*	*	*																			
29	MOVE JK-ITO TO DB-ITO			*	*																								
30	MOVE JK-SEC TO DB-SEC			*	*																								
31	MOVE JK-SEC TO FE-SEC			*	*																								
32	MOVE JK-ID TO DB-ID			*	*																								
33	MOVE JK-ID TO FE-ID			*	*																								
34	MOVE W71-SEM TO DB-SEM			*	*																								
35	MOVE W71-SEM TO FE-SEM			*	*																								
36	MOVE SPACE TO DB-RESTE,W1-ZONB			*	*																								
37	PERFORM 3A15 THRU 3A20			*	*																								
38	MOVE 1 TO W11-ER (6)										*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
	SOUS-TABLE DE DEGRE 1																												SOUS-TABLE DE DEGRE 1
39	MOVE SPACE TO W1-ZONB					*			*																				
40	MOVE W1-ZONA TO DB-ENR					*	*		*	*																			
41	MOVE W1-ID TO DB-ID					*	*		*	*																			
42	MOVE 1 TO W11-ER (1)										*	*	*	*	*	*	*	*	*										
43	MOVE 0 TO W11-ER (1)																				*								
44	MOVE ZERO TO W11-ER (1)																					*	*						
45	MOVE 1 TO W11-ER (10)																					*	*						
46	MOVE 1 TO AIG4																							*	*				
47	MOVE C TO AIG4																										*	*	
48	PERFORM ECRIFAU																				*	*	*	*	*	*	*	*	
49	MOVE JK-ITO TU DB-ITO																					*	*						
50	MOVE JK-SEC TO DB-SEC																					*	*						
51	MOVE JK-SEC TO FE-SEC																					*	*						
52	MOVE JK-ID TO DB-ID																					*	*						
53	MOVE JK-ID TO FE-ID																					*	*						
54	MOVE W71-SEM TO DB-SEM					*	*		*	*												*	*						
55	MOVE W71-SEM TO FE-SEM																					*	*						
56	MOVE SPACE TO DB-RESTE,W1-ZONB																					*	*						
57	PERFORM 3A15 THRU 3A20																					*	*						
	SOUS-TABLE DE DEGRE 2																												SOUS-TABLE DE DEGRE 2
58	MOVE 0 TO W11-ER (1)											*																	
59	MOVE ZERO TC W11-ER (1)												*	*															
60	MOVE 1 TO W11-ER (10)												*	*															
61	MOVE 1 TO AIG4														*	*													
62	MOVE 0 TO AIG4																	*	*										
63	PERFORM ECRIFAU											*	*	*	*	*	*	*	*										
64	MOVE JK-ITO TO DB-ITO												*	*															
65	MOVE JK-SEC TO DB-SEC												*	*															
66	MOVE JK-SEC TO FE-SEC												*	*															
67	MOVE JK-ID TO DB-ID												*	*															
68	MOVE JK-ID TO FE-ID												*	*															
69	MOVE SPACE TO W1-ZONB																							*			*		
70	MOVE W1-ZONA TO DB-ENR																							*	*		*	*	
71	MOVE W1-ID TO DB-ID																							*	*		*	*	
72	MOVE W71-SEM TO DB-SEM												*	*										*	*		*	*	
73	MOVE W71-SEM TO FE-SEM												*	*															
74	MOVE SPACE TO DB-RESTE,W1-ZONB												*	*															
75	PERFORM 3A15 THRU 3A20												*	*															
	SOUS-TABLE DE DEGRE 3																												SOUS-TABLE DE DEGRE 3
76	MOVE SPACE TO W1-ZONB														*			*											
77	MOVE W1-ZONA TO DB-ENR														*	*		*	*										
78	MOVE W1-ID TO DB-ID														*	*		*	*										
79	MOVE W71-SEM TO DB-SEM														*	*		*	*										
	RUPTURES DE SEQUENCE																												RUPTURES DE SEQUENCE
80	GO TO 2A10	*									*									*									
81	GO TO 2A20		*					*				*					*				*					*			
82	GO TO 2A09			*	*	*	*		*	*			*	*	*	*		*	*			*	*	*	*		*	*	

```
                                        * TABLE OUVERTE 2A09 *
                                        **********************

  TABLE NUMERO  12

   REGLES   1   A     2
  ------------------------------------------II---I---I
                                            II   I   I
1  DB-CLE VALID KEY ....................... II O I N I
                                            II   I   I
  ------------------------------------------II---I---I
  ------------------------------------------II---I---I
2 WRITE    DB-ENR ......................... II * I   I
                                            II   I   I
3 MOVE     W1-SEM TO DB-SEM ............... II   I * I
                                            II   I   I
4 MOVE     W1-ID TO DB-ID ................. II   I * I
                                            II   I   I
5 PERFORM  2A15 THRU 1 .................... II   I * I
                                            II   I   I
6 GO       TO 2A20 ........................ II   I * I
                                            II   I   I
7 DISPLAY  'IK 2A09 MATRICULE DEJA ECRIT'  II   I * I
                                            II   I   I
  ------------------------------------------II---I---I
                                 RUPTURES DE SEQUENCE
                                            II   I   I
8 GO       TO 2A20 ........................ II * I * I
                                            II   I   I
  ------------------------------------------II---I---I
```

```
                                  * TABLE OUVERTE 2A10 *
                                  **********************

  TABLE NUMERO  13

   REGLES   1   A     2
  ----------------------------II---I---I
                              II   I   I
  ----------------------------II---I---I
                              SOUS-TABLE DE DEGRE   1
                              II   I   I
1  DB-CLE VALID KEY ......... II O I N I
                              II   I   I
  ----------------------------II---I---I
  ----------------------------II---I---I
2 MOVE      W1-ZONA TO DB-ENR  II * I * I
                              II   I   I
3 MOVE      W1-ID TO DB-ID ... II * I * I
                              II   I   I
4 MOVE      W1-SEM TO DB-SEM . II * I * I
                              II   I   I
  ----------------------------II---I---I
                              SOUS-TABLE DE DEGRE   1
                              II   I   I
5 WRITE     DB-ENR ........... II * I   I
                              II   I   I
6 MOVE      W1-SEM TO DB-SEM . II   I * I
                              II   I   I
7 MOVE      W1-ID TO DB-ID ... II   I * I
                              II   I   I
  ----------------------------II---I---I
                              RUPTURES DE SEQUENCE
                              II   I   I
8 GO        TO 2A20 .......... II * I   I
                              II   I   I
9 GO        TO 2A15 .......... II   I * I
                              II   I   I
  ----------------------------II---I---I
```

```
                                        * TABLE OUVERTE 2A15 *
                                        **********************

  TABLE NUMERO  14

   REGLES   1   A     2
  ------------------------------II---I---I
                                II   I   I
1  DB-CLE VALID KEY ..........  II O I N I
                                II   I   I
  ------------------------------II---I---I
  ------------------------------II---I---I
2 READ      DB-CARTBON RECORD   II * I   I
                                II   I   I
  ------------------------------II---I---I
                                RUPTURES DE SEQUENCE
                                II   I   I
3 GO        TO 1 ..............  II * I   I
                                II   I   I
4 STOP      RUN ...............  II   I * I
                                II   I   I
  ------------------------------II---I---I
```

```
                                        * TABLE FERMEE 1 *
                                        ******************

  TABLE NUMERO  15

   REGLES   1   A     3
  ---------------------------------II---I---I---I
                                   II   I   I   I
1  DB-RESTE = SPACE ..............  II O I O I N I
                                   II   I   I   I
2  DB-CLE VALID KEY ..............  II O I N I   I
                                   II   I   I   I
  ---------------------------------II---I---I---I
  ---------------------------------II---I---I---I
3 MOVE      W1-ZONB TO DB-RESTE    II * I * I   I
                                   II   I   I   I
4 REWRITE   DB-ENR ................ II * I   I   I
                                   II   I   I   I
  ---------------------------------II---I---I---I
                                  RUPTURES DE SEQUENCE
                                   II   I   I   I
5 EXIT      1 ..................... II * I   I * I
                                   II   I   I   I
6 GO        TO 1 .................. II   I * I   I
                                   II   I   I   I
  ---------------------------------II---I---I---I
```

```
                                        * TABLE OUVERTE 2A20 *
                                        **********************

  TABLE NUMERO  16

   REGLES   1   A     3
  ----------------------------II---I---I---I
                              II   I   I   I
1  AIGEND = 1 ..............  II O I O I N I
                              II   I   I   I
2  AIG2 = 0 ................  II O I N I   I
                              II   I   I   I
  ----------------------------II---I---I---I
  ----------------------------II---I---I---I
3 MOVE      1 TO I ..........  II * I * I * I
                              II   I   I   I
4 MOVE      0 TO W75-CONTSE .  II * I * I * I
                              II   I   I   I
5 MOVE      0 TO AIG4 .......  II * I * I * I
                              II   I   I   I
6 MOVE      0 TO W1-ERR .....  II * I * I * I
                              II   I   I   I
7 MOVE      0 TO FE-ERIND ...  II * I * I * I
                              II   I   I   I
8 MOVE      ZERO TO W1-ERR ..  II * I * I * I
                              II   I   I   I
9 MOVE      SPACE TO W1-ZONA   II * I * I * I
                              II   I   I   I
  ----------------------------II---I---I---I
                               RUPTURES DE SEQUENCE
                              II   I   I   I
10 GO       TO 1A15 .........  II * I   I   I
                              II   I   I   I
11 STOP     RUN .............  II   I * I   I
                              II   I   I   I
12 GO       TO DEB ..........  II   I   I * I
                              II   I   I   I
  ----------------------------II---I---I---I
```

```
                                        * TABLE FERMEE MATMANQ *
                                        **************************

  TABLE NUMERO  17

   REGLES   1   A     3
  ----------------------------II---I---I---I
                              II   I   I   I
  ----------------------------II---I---I---I
                              SOUS-TABLE DE DEGRE   1
                              II   I   I   I
1  DB-CLE VALID KEY ......... II O I O I N I
                              II   I   I   I
  ----------------------------II---I---I---I
                              SOUS-TABLE DE DEGRE   2
                              II   I   I   I
2  DB-RESTE = SPACE ......... II O I N I   I
                              II   I   I   I
  ----------------------------II---I---I---I
  ----------------------------II---I---I---I
3 MOVE      W71-SEM TO DB-SEM  II * I * I * I
                               II   I   I   I
4 MOVE      JK-ID TO DB-ID ... II * I * I * I
                               II   I   I   I
5 MOVE      JK-SEC TO DB-SEC . II * I * I * I
                               II   I   I   I
6 MOVE      JK-ITO TO DB-ITO . II * I * I * I
                               II   I   I   I
  ----------------------------II---I---I---I
                              SOUS-TABLE DE DEGRE   1
                              II   I   I   I
7 READ      DB-CARTBON RECORD  II * I * I   I
                               II   I   I   I
8 MOVE      SPACE TO DB-RESTE  II   I   I * I
                               II   I   I   I
9 MOVE      W71-SEM TO FE-SEM  II   I   I * I
                               II   I   I   I
10 MOVE     JK-ID TO FE-ID ... II   I   I * I
                               II   I   I   I
11 MOVE     JK-SEC TO FE-SEC . II   I   I * I
                               II   I   I   I
  ----------------------------II---I---I---I
                              SOUS-TABLE DE DEGRE   2
                              II   I   I   I
12 MOVE     W71-SEM TO FE-SEM  II * I   I   I
                               II   I   I   I
13 MOVE     JK-ID TO FE-ID ... II * I   I   I
                               II   I   I   I
14 MOVE     JK-SEC TO FE-SEC . II * I   I   I
                               II   I   I   I
  ----------------------------II---I---I---I
                              RUPTURES DE SEQUENCE
                              II   I   I   I
15 GO       TO 3A15 .......... II * I   I * I
                               II   I   I   I
16 EXIT     3A20 ............. II   I * I   I
                               II   I   I   I
  ----------------------------II---I---I---I
```

```
                                   * TABLE FERMEE 3A15 *
                                   **********************

  TABLE NUMERO  18

   REGLES   1   A     1
  ------------------------------II---I
                                 II   I
1 MOVE     ZERO TO FE-ERIND .   II * I
                                 II   I
2 MOVE     1 TO FE-ERR (1) ..   II * I
                                 II   I
3 MOVE     SPACE TO FE-RESTE    II * I
                                 II   I
4 WRITE    FE-ENR ...........   II * I
                                 II   I
  ------------------------------II---I
                                 RUPTURES DE SEQUENCE
                                 II   I
5 EXIT     3A20 .............   II * I
                                 II   I
  ------------------------------II---I
```

```
                                * TABLE FERMEE ECRIFAU *
                                **********************

  TABLE NUMERO  19

   REGLES   1   A     1
  ------------------------------II---I
                                II   I
1 MOVE      W1-ZONAC TO FE-ENR   II * I
                                II   I
2 WRITE     FE-ENR ............ II * I
                                II   I
  ------------------------------II---I
                                RUPTURES DE SEQUENCE
                                II   I
3 EXIT      ECRIFAU ........... II * I
                                II   I
  ------------------------------II---I
```

```
XQT TABOL.PD3C


  1

CT,  1, CREATION D'UN FICHIER RESUME DES CONTRATS - OUVERTURE , * ,1.

IGT, CREATION-FICHIER-REDUIT.


  2

IPR,1/2.

EG,T, ENREGISTREMENT-RESUME.

CT,5 / 6 , L'ENREG RESUME CONTIENT LE NUMERO D'AFFAIRE  ET LA DATE FIN DE
        CONTRAT, * , 1/2.

CT,4 , LIRE UN ENREGISTREMENT DU FICHIER COMPLET DES AFFAIRES , *, 1/2/ 3.

ND, 1/2 , L'ENREG SUIVANT DU FICHIER COMPLET CONCERNE UNE AFFAIRE , O/N/FIN
        , 1, 3 , 4 .

ND, 3 , ON PEUT ECRIRE L'ENREGISTREMENT RESUME , O /N , 1, 2.

CT, 8/9, SIGNALER  UNE ERREUR, *, 2.

UL.


  3

RS,1.

ND, 1 , LE FICHIER DES CARTES DE POINTAGES EST VIDE, N/O , 1,2.

CT, 2/3/4/5/6 , OUVRIR LES FICHIERS , * ,  1/2.

CT, 7/8, LIRE LA PREMIERE CARTE DE POINTAGE , * , 1.

CT, 9, SIGNALER L'ERREUR , *, 2.


  4

ND, 1/2 , LE MATRICULE DE LA DERNIERE CARTE POINTAGE 1-FAUX 2-BON 3-DOUBLE
       1/2/3 , 1,2,3.

ND, 3/4,  LE FICHIER PERSONNEL A ETE LU  +- LOIN QUE LE FICHIER POINTAGE,
       +/-/= ,  3,4,5.
```

```
CT, 5/6/7/8/9/10/11, RAPPELER DANS CHAQUE CAS LE TYPE D'ERREUR , *, 1/3/4/5.

CT, 12, METTRE LA CARTE DE POINTAGE EN MEMOIRE , *, 1/3/5.

  5

ONO, 1, LE MATRICULE INDIQUE SUR LA DERNIERE CARTE ETAIT FAUX , O / N,  1, 4

ONO,2/3, LE FICHIER PERSONNEL A ETE LU +- LOIN QUE LE FICHIER POINTAGE,
        +/-/= , 1,2,3.

HGT, COMPARAISON-DES-MATRICULES.

CT, 4/5/6/7/8/9 , RAPPELER DANS CHAQUE CAS LE TYPE D'ERREUR, *, 1/2/3 .

  6

ONO,1,IL RESTE DES CARTES A LIRE., O / N,  1 , 7 .

ONO,2,            LA CARTE EST L'UNE DES TROIS PREMIERES , O/N , 1,2.

ONO,3,            LA CARTE EST L'UNE DES TROIS PREMIERES , O/N , 2,3.

ONO,4,            LA CARTE EST L'UNE DES TROIS PREMIERES , O/N , 3,4  .

ONO,5/6, LE FICHIER POINTAGE A ETE LU +- LOIN QUE LE FICHIER PERSONNEL,
        +/-/=,4,5,6.

HGT,LECTURE-PREMIERES-CARTES.

CT,7,LIRE LA CARTE SUIVANTE, *,1.

CT,8/9/10, INITIALISER LE TRAITEMENT SUIVANT,*,7.

CT,11, RAPPELER LA CARTE EN MEMOIRE,*,7.

CT,12/13/14/15/16, DANS CHAQUE CAS RAPPELER L'ERREUR ,*,4/5.

CT,17,RAPPELER LA CONCORDANCE,*,6.

CT,18, RAPPELER LA CARTE EN MEMOIRE ,*,4/6.
```

* TABLE OUVERTE CREATION-FICHIER-REDUIT *

****

TABLE NUMERO 1

REGLES 1 A 4					
	TABLE ENREGISTREMENT-RESUME				
1 L'ENREG SUIVANT DU FICHIER COMPLET CONCERNE UNE AFFAIRE	O	I	O	I	N
2 ON PEUT ECRIRE L'ENREGISTREMENT RESUME	O	I	N	I	
3 CREATION D'UN FICHIER RESUME DES CONTRATS - OUVERTURE	*	I	*	I	*
	TABLE ENREGISTREMENT-RESUME				
4 L'ENREG RESUME CONTIENT LE NUMERO D'AFFAIRE ET LA DATE FIN DE CONTRAT	*	I	*	I	
5 WRITE MINI-ENR	*	I		I	
6 SIGNALER UNE ERREUR		I	*	I	
RUPTURES DE SEQUENCE					
7 GO TO ENREGISTREMENT-RESUME	*	I	*	I	*
8 GO TO INIT		I		I	*

```
                    * TABLE OUVERTE INIT *
                    **********************

 TABLE NUMERO   2

  REGLES   1   A     2
 ------------------------------------------------------II---I---I
                                                        II   I   I
 ------------------------------------------------------II---I---I
                          SOUS-TABLE DE DEGRE   1
                                                        II   I   I
1  LE FICHIER DES CARTES DE POINTAGE EST VIDE ..... II N I O I
                                                        II   I   I
 ------------------------------------------------------II---I---I
 ------------------------------------------------------II---I---I
2 OUVRIR   LES FICHIERS ............................... II * I * I
                                                        II   I   I
 ------------------------------------------------------II---I---I
                          SOUS-TABLE DE DEGRE   1
                                                        II   I   I
3 LIRE     LA PREMIERE CARTE DE POINTAGE ..... II * I   I
                                                        II   I   I
4 SIGNALER L'ERREUR ................................... II   I * I
                                                        II   I   I
 ------------------------------------------------------II---I---I
                          RUPTURES DE SEQUENCE
                                                        II   I   I
5 GO       TO DEB ..................................... II * I   I
                                                        II   I   I
6 STOP     RUN ........................................ II   I * I
                                                        II   I   I
 ------------------------------------------------------II---I---I
```

* TABLE OUVERTE DEB *
****

TABLE NUMERO 3

	REGLES 1 A 5									
1	LE MATRICULE DE LA DERNIERE CARTE POINTAGE 1-FAUX 2-BON 3-DOUBLE	1		2		3		3		3
2	LE FICHIER PERSONNEL A ETE LU +- LOIN QUE LE FICHIER POINTAGE					+		-		=
3	RAPPELER DANS CHAQUE CAS LE TYPE D'ERREUR	*				*		*		*
4	METTRE LA CARTE DE POINTAGE EN MEMOIRE	*				*				*
	RUPTURES DE SEQUENCE									
5	GO TO 1B16	*				*				*
6	GO TO COMPARAISON-DES-MATRICULES			*				*		

* TABLE OUVERTE COMPARAISON-DES-MATRICULES *
****

TABLE NUMERO 4

REGLES 1 A 4

		1	2	3	4
1	LE MATRICULE INDIQUE SUR LA DERNIERE CARTE ETAIT FAUX	O	O	O	N
2	LE FICHIER PERSONNEL A ETE LU +- LOIN QUE LE FICHIER POINTAGE	+	-	=	
3	RAPPELER DANS CHAQUE CAS LE TYPE D'ERREUR	*	*	*	
4	MOVE AC-ENR TO W1-ZONA	*		*	
	RUPTURES DE SEQUENCE				
5	GO TO 1B16	*		*	
6	GO TO COMPARAISON-DES-MATRICULES		*		
7	GO TO LECTURE-PREMIERES-CARTES				*

* TABLE OUVERTE LECTURE-PREMIERES-CARTES *

TABLE NUMERO 5

	REGLES 1 A 5									
1	IL RESTE DES CARTES A LIRE	O	O	O	O	N				
	SOUS-TABLE DE DEGRE 1									
2	LA CARTE EST L'UNE DES TROIS PREMIERES	O	N	N	N					
3	LE FICHIER POINTAGE A ETE LU +- LOIN QUE LE FICHIER PERSONNEL		+	-	=					
4	LIRE LA CARTE SUIVANTE	*	*	*	*					
5	INITIALISER LE TRAITEMENT SUIVANT					*				
6	RAPPELER LA CARTE EN MEMOIRE					*				
	SOUS-TABLE DE DEGRE 1									
7	DANS CHAQUE CAS RAPPELER L'ERREUR		*	*						
8	RAPPELER LA CONCORDANCE				*					
9	RAPPELER LA CARTE EN MEMOIRE		*							
	RUPTURES DE SEQUENCE									
10	GO TO LECTURE-PREMIERES-CARTES	*								
11	GO TO 1B16		*		*	*				
12	GO TO COMPARAISON-DES-MATRICULES			*						

Dernière page d'une thèse

VU

Grenoble, le 26 Octobre 1973

Le Président de la thèse

Vu, et permis d'imprimer,

Grenoble, le 8 décembre 1973

Le Président de l'Université
Scientifique et Médicale